Data Mining und Maschinelles Lernen

Prof. Kristian Kersting
Zhongjie Yu
Johannes Czech

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 2021
24. Juli 2021
Bonusübungsblatt 1

Die **Abgabefrist** dieser Bonusübung ist am **31.07.2021** um **23:59 Uhr**.

Aufgabe	1	2	3	4	5
Maximal Punktzahl	12	10	18	12	18
Erreichte Punktzahl					

Gruppe AE	**Nachnahme**	**Vorname**	**Matrikelnummer**
1	Liu	Lanmiao	2571210
2	Shao	Yuening	2504740
3	Li	Zongjian	2595681
4	Liu	Yue	2803140

Benötigte Dateien
Die benötigten Datensatz zu Aufgabe 1 und 2 sowie Skriptvorlagen finden Sie in unserem Moodle-Kurs:
`https://moodle.informatik.tu-darmstadt.de/course/view.php?id=1058`

Den Datensatz zu Aufgabe 3, 4 und 5 finden Sie auf `kaggle.com`:
`https://www.kaggle.com/slothkong/10-monkey-species`

Gruppeneinteilung
Bearbeiten Sie diese Übung in Dreier- oder Vierergruppen. Es steht Ihnen frei die Gruppen selbst zu bilden. Nutzen Sie hierfür die Gruppeneinteilung und das Forum in Moodle. Sehen Sie bitte aufgrund der aktuellen Coronakrise davon ab, sich lokal zu treffen und nutzen Sie stattdessen digitale Kommunikationskanäle. Zur gemeinsamen Bearbeitung der Abgabe können sie beispielsweise `https://overleaf.com` nutzen.

Theoretische Aufgaben
Bei theoretischen Übungsaufgaben, sind wir Ihnen sehr dankbar, wenn Sie diese in LaTeX formatieren und als PDF einreichen. Nutzen Sie hierfür die LaTeX-Vorlage und die vorgesehene Blöcke:

```
\begin{solution}
% your solution goes here
\end{solution}
```

Geben Sie dabei ihre Gruppenmitglieder und Gruppennummber in der Datei `group_members.tex` an.

Wenn Sie mit LaTeX nicht ausreichend vertraut sind, können Sie auch einen hochauflösenden Scan einer handgeschriebenen Lösung einreichen. Bitte schreiben Sie ordentlich und leserlich.

Programmieraufgaben
Bei Aufgaben, die mit einem **</>** versehen sind, handelt es sich um Programmieraufgaben. Bearbeiten Sie in diesem Fall die vorgegebene Programmiervorlage. Verwenden Sie bevorzugt $\geq$ **Python 3.7**, da wir diese Version zum Testen ihrer Lösung benutzen. Benennen Sie die Funktionsdateien nicht um und ändern Sie die angegebenen Funktionssignaturen nicht. Wenn Sie das Gefühl haben, dass es ein Fehler bei den Zuweisungen, fragen Sie uns auf Moodle.

Formalien zur Abgabe
Bitte laden Sie Ihre Lösungen in der entsprechenden Rubrik auf Moodle hoch. Sie müssen **nur eine Lösung pro Gruppe** einreichen. Wenn Sie keinen Zugang zu Moodle haben, setzen Sie sich bitte so schnell wie möglich mit uns in Verbindung. Laden Sie alle Ihre Lösungsdateien als eine einzige .zip-Datei hoch. Bitte beachten Sie, dass wir keine anderen als die angegebenen Dateiformate akzeptieren. Laden Sie **nicht den gegeben Datensatz zur Programmieraufgabe** und **nicht die trainierten Modelle** in der Abgabe hoch.

Nutzen Sie folgende Namensgebung:

```
dmml_bonus_group<groupid>.zip
├── dmml_bonus_2021.pdf
├── 01_ptu_classification.py
├── 02_ptu_regression.py
├── 03_monkey_traditional_ml.py
├── 04_monkey_simple_cnn.ipynb
├── 05_monkey_transfer_cnn.ipynb
├── utils.py
└── requirements.txt
```

Geben Sie die Jupyter Notebooks **inklusive der Ausgabe aller Zellen** ab und geben Sie in `requirements.txt` die Liste der benötigten Bibliotheken an. In `utils.py` können Sie eigene Funktionen definieren, die Sie in mehreren Skripten nutzen.

Verspätete Abgaben
Verspätete Abgaben werden akzeptiert, aber für jeden Tag, an dem die Abgabefirst überschritten wird, werden 25 % der insgesamt erreichbaren Punkte abgezogen. Am fünften Tag nach Abgabefrist, können Sie die Aufgabe nicht mehr einreichen.

Bewertungsfaktoren
Die Bewertung dieser Übung hängt von den folgenden Faktoren ab:

- Richtigkeit der Antwort
- Klarheit der Präsentation der Ergebnisse
- Schreibstil

Wenn Sie bei einer Aufgabe nicht weiterkommen, versuchen Sie zu erklären warum und beschreiben Sie die Probleme, auf die Sie gestoßen sind, da Sie dafür Teilpunkte erhalten können.

Umgang mit Plagiaten
Sie dürfen gerne kursbezogene Themen in der Vorlesung oder in unseren Moodle Foren diskutieren. Sie sollten allerdings keine Lösungen mit anderen Gruppen teilen, und alles, was Sie einreichen, muss Ihre eigene Arbeit sein. Es ist Ihnen auch nicht gestattet, Material aus dem Internet zu kopieren. Sie sind verpflichtet, jede Informationsquelle, die Sie zur Lösung der Übungsaufgabe verwendet haben (d.h. andere Materialien als die Vorlesungsmaterialien), anzuerkennen. Zitierungen haben keinen Einfluss auf Ihre Note. Mindestens 60 % der abgegeben Lösung für individuelle Aufgaben muss selbst formuliert sein. Nicht anerkennen einer Quelle, die Sie verwendet haben, ist dagegen ein klarer Verstoß gegen die akademische Ethik. Beachten Sie, dass die Universität sehr ernsthaft mit Plagiaten umgeht.

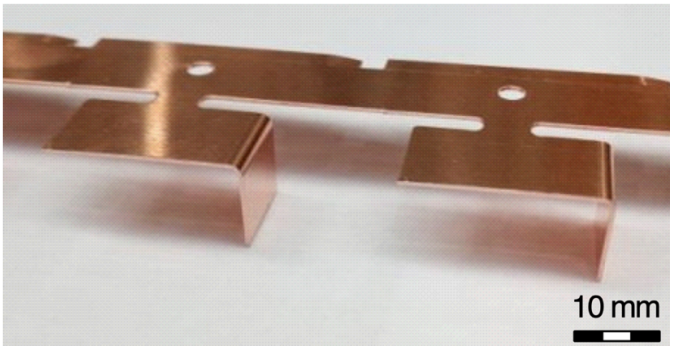
Anrechnung der Bonuspunkte
Die Klausur wird eine maximale Punktzahl von 100 Punkten haben. In dieser Übung können Sie bis zu **70 Punkte** erreichen. Die angerechneten Bonuspunkte für die Klausur ergeben sich über folgende Formel:

$$\text{Bonuspunkte für Klausur} = \frac{\text{erreichte Bonuspunkte}}{10.0}$$

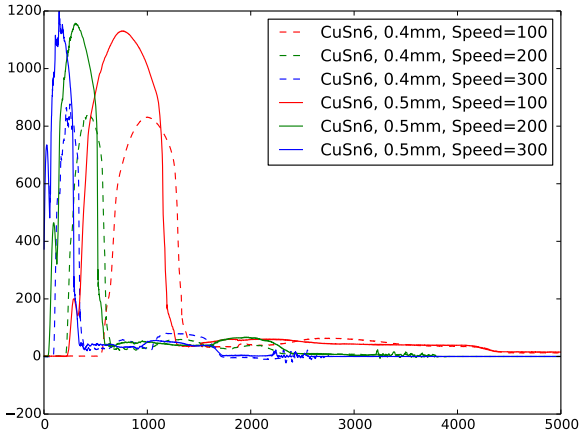
Das Interval von fünf Punkten in der Klausur wird etwa einer Notenstufe entsprechen.

Aufgabe 1.1: </> Entscheidungsbäume - Klassifikation (12)

Im Institut Institut für Produktionstechnik und Umformmaschinen, kurz PtU[1], der TU Darmstadt gibt es die Aufgabe eines Scherschneideverfahrens. Dabei wird mit Hilfe eines Stempels ein Loch in einen metallischen Werkstoff gestanzt, siehe Abbildung 1. Es gibt zwei Werkstoffe im Versuch: `CuSn6` und `16MnCr5`. Die Dicke des Materials beträgt entweder $0.4\,\text{mm}$ oder $0.5\,\text{mm}$. Die Geschwindigkeit des Stempels liegt in den drei Stufen 100, 200 und 300 vor.



(a) Produziertes Werkstück.



(b) Beispielverlauf des Kraftsignals.

Abbildung 1: Veranschaulichung des PtU Prozesses.

Es gibt mehrere Sensoren, die den Status des Stanzen messen:

- Kraft Sensor / Force sensor (Fz)
- Verschiebungssensor / Displacement sensor (w)
- Beschleunigungssensor / Acceleration sensor (acc)

Im Experiment messen die Sensoren den Status des Stanzen von Beginn bis Ende jedes Prozesses. Die Zeitreihendaten von jedem Sensor werden als 1D-Vektor dargestellt. Das Interessante an dieser Aufgabe ist, dass der Status des Stanzen von der Art und Dicke des Materials abhängt. Andererseits ist es uns möglich, aus den Zeitreiheninformationen von den Sensoren auf die Art und Dicke des Materials und die Geschwindigkeit des Stanzen zu schließen.

- **`Filename_Fz_raw.csv`**: Enthält Daten aus dem Kraftsensor. Jeder Zeilenvektor repräsentiert das Kraftsignal. Die Anzahl der Zeilen in der Datei beträgt 2787, was der Anzahl der Experimente im Datensatz entspricht.
- **`Filename_Speed.csv`**: Enthält die Geschwindigkeit des Schlags der 2787 Proben. Mögliche Werte für Geschwindigkeit sind dabei $\{100, 200, 300\}$.
- **`Filename_thickness.csv`**: Enthält die Materialstärke in Millimeter der 2787 Proben.

In dieser Aufgabe verwenden wir die Daten des Kraftsensors, um die Proben nach der Geschwindigkeit des Stempels zu klassifizieren.

1.1a) </> `01_ptu_classification.py` (9 Punkte)

Laden Sie die Daten des Kraftsensor aus `Filename_Fz_raw.csv` und die Geschwindigkeit des Stanzen aus `Filename_Speed.csv` und vervollständigen Sie den Code in `01_ptu_classification.py`:

[1] `https://www.ptu.tu-darmstadt.de/`

</> Visualisieren Sie die ersten zehn Zeitreihen des Trainingsdatensatzes als Linien-Plot in der Methode `visualize_data()`. Stellen Sie dabei die unterschiedlichen Klassen farblich dar und geben Sie eine Legende an. Vergessen Sie nicht die Achsen zu beschriften und vermeiden Sie doppelte Einträge in der Legende. (2 Punkte)

</> Trainieren Sie einen Entscheidungsbaum zur Klassifikation in der Methode `fit_dt_classifier()` mithilfe von `sklearn` unter der Verwendung der Standardparameter. Der Entscheidungsbaum darf allerdings nur eine maximale Tiefe von 3 erreichen. (1 Punkt)

</> Ermitteln Sie die Testgenauigkeit in der Methode `get_test_accuracy()`. (1 Punkt)

</> Plotten Sie den resultierenden Entscheidungsbaum in `export_tree_plot()`. Sie können dabei die Funktion `tree.export_graphviz()` verwenden. (1 Punkt)

</> Führen Sie eine Grid-Suche für einen AdaBoost-Klassifier in der Methode `run_grid_search()` durch. Verwenden Sie dabei pro Konfiguration eine 5-fache Kreuzvalidierung. Der Suchraum beschränkt sich dabei auf die Hyperparameter: Maximale Baumtiefe der Basislerner (Entscheidungsbäume), Lernrate und Anzahl an Estimatoren. Sie können dabei die Funktion `sklearn.model_selectionGridSearchCV()` verwenden. Geben Sie zuletzt die Konfiguration der als best gefundenen Hyperparameter zurück. (3 Punkte)

</> Trainieren Sie in der Methode `fit_ada_boost()` einen AdaBoost-Klassifier auf den Trainingsdatensatz mit der als best gefundenen Hyperparamter-Konfiguration. (1 Punkt)

```python
from sklearn import tree
from sklearn.model_selection import train_test_split
import numpy as np
import graphviz
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from util import get_x_down_sampled
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import make_scorer
from sklearn.model_selection import GridSearchCV
import pydotplus
from collections import OrderedDict

def fit_dt_classifier(X_train: np.ndarray, y_train: np.ndarray, max_depth: int = None) -> tree.
    DecisionTreeClassifier:
    """Creates and fits a decision tree classifier on the training data."""

    # max_depth:The maximum depth of the tree
    # Use sklearn.tree.DecisionTreeClassifier()
    classifier=tree.DecisionTreeClassifier(max_depth=max_depth)

    # fit():Build a decision tree classifier from the training set(X_train, y_train)
    classifier=classifier.fit(X_train,y_train)

    return classifier


def get_test_accuracy(clf, X_test: np.ndarray, y_test: np.ndarray) -> float:
    """Evaluates the test accuracy for a given classifier and test dataset."""

    acc=clf.score(X_test,y_test)
    return acc


def export_tree_plot(clf, filename: str) -> None:
    """Exports the tree plot for the given classifier as a pdf with given filename."""

    # Export the decision tree classifier process as a graphviz.dot file.
    dot_data=tree.export_graphviz(clf)

    # A Dot class will be returned to represent the graph
    graph=pydotplus.graph_from_dot_data(dot_data)

    # Generate PDF file
    graph.write_pdf(filename)
```

```python


def visualize_data(X_train: np.ndarray, y_train: np.ndarray, nb_samples_to_plot: int) -> None:
    """Visualizes the first nb_samples_to_plot in a plot with a legend of the class and exports the plot as a pdf."""

    # nb_samples_to_plot=10
    X_train_plot=X_train[:nb_samples_to_plot]
    y_train_plot=y_train[:nb_samples_to_plot]

    fig,ax=plt.subplots()
    t=np.linspace(0,5000,X_train.shape[1])
    # Xaxis:t, Yaxis:=X_train[:10], true_label:y_train[:10]
    for Y,true_label in zip(X_train_plot,y_train_plot):
        if true_label==100:
            ax.plot(t,Y,c='r',label='Speed=100')
        elif true_label==200:
            ax.plot(t,Y,c='g',label='Speed=200')
        else:
            ax.plot(t,Y,c='b',label='Speed=300')

    # Return handles and labels for legend
    handles,labels=ax.get_legend_handles_labels()

    # Avoid duplicate entries in the legend
    by_label=OrderedDict(zip(labels,handles))
    ax.legend(by_label.values(),by_label.keys(),loc='best')

    ax.set_title('first 10 time series of the training data')
    plt.xlabel("T",fontsize=20)
    plt.ylabel("Force",fontsize=20)
    plt.savefig('./speed_data.pdf')
    plt.show()


def run_grid_search(X_train, y_train, max_depth_range, learning_rate_range, n_estimators_range) -> (int, float, int):
    """Runs a grid search on the training data using the specified hyperparameter ranges using
     a 5 fold cross-valdidation per configuration. At last, the best hyperparamter tuple is returned."""

    df_clf=tree.DecisionTreeClassifier()

    parameters={'base_estimator__max_depth':max_depth_range,'learning_rate':learning_rate_range,'n_estimators':n_estimators_range}
    Ada_clf=AdaBoostClassifier(df_clf)

    # Exhaustive search over specified parameter values for an estimator.
    clf=GridSearchCV(Ada_clf,parameters,n_jobs=-1)
    clf.fit(X_train,y_train)

    return clf.best_params_['base_estimator__max_depth'], clf.best_params_['learning_rate'], clf.best_params_['n_estimators']


def fit_ada_boost(X_train: np.ndarray, y_train: np.ndarray, max_depth: int, learning_rate: float, n_estimators: int) -> AdaBoostClassifier:
    """Creates and fits an ada boost classifier on the training data."""
    # Use the best hyperparameters(max_depth,learning_rate,n_estimators) obtained through grid search
    df_clf=tree.DecisionTreeClassifier(max_depth=max_depth)

    Ada_clf=AdaBoostClassifier(base_estimator=df_clf,learning_rate=learning_rate,n_estimators=n_estimators)

    Ada_clf.fit(X_train,y_train)
    return Ada_clf


def main():
    # for reproducibility
    np.random.seed(42)
```

```
    # load data
    X_data = get_x_down_sampled('./PtU/FileName_Fz_raw.csv')
    y_data = np.loadtxt(open('./PtU/FileName_Speed.csv', 'r'), delimiter=",", skiprows=0)

    print("X_data.shape:", X_data.shape)
    print("y_data.shape:", y_data.shape)
    print("X_sample.shape:", X_data.shape)

    # split training and test sets
    X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.2, random_state=42)

    # visualize the task
    visualize_data(X_train, y_train, 10)

    # train
    clf = fit_dt_classifier(X_train, y_train, max_depth=3)
    # predict
    acc = get_test_accuracy(clf, X_test, y_test)
    print('Test Accuracy:', acc)

    print("predict_proba:", clf.predict_proba(X_test))

    # plot tree
    export_tree_plot(clf, "classification_tree.pdf")

    # run grid search
    max_depth_range = list(range(1, 5))
    learning_rate_range = [2 ** i for i in range(-2, 2)]
    n_estimators_range = [2 ** i for i in range(5, 8)]
    best_max_depth, best_lr, best_n_estimators = run_grid_search(X_train, y_train,  max_depth_range,
                                                                 learning_rate_range, n_estimators_range)
    clf = fit_ada_boost(X_train, y_train, best_max_depth, best_lr, best_n_estimators)
    print(f'Best configuration: max_depth: {best_max_depth}, lr: {best_lr}, n_estimators: {
    best_n_estimators}')

    # predict
    y_pred = clf.predict(X_test)
    print("y_pred:", y_pred[:10], "...")
    print("y_test:", y_test[:10], '...')

    # show confusion matrix
    print("Train Confusion Matrix for Ada Boost Classifier:\n", confusion_matrix(y_train, clf.predict(
    X_train)))
    print("Test Confusion Matrix for Ada Boost Classifier:\n", confusion_matrix(y_test, y_pred))

    # evaluate
    acc = accuracy_score(y_test, y_pred)
    print('Ada Boost Test Accuracy:', acc)


if __name__ == '__main__':
    main()
```

1.1b) Datenvisualisierung (1 Punkt)

Geben Sie die erstellte Visualsierung der Daten aus Unteraufgabe a) an.

Lösungsvorschlag:

s. Abb. 2.

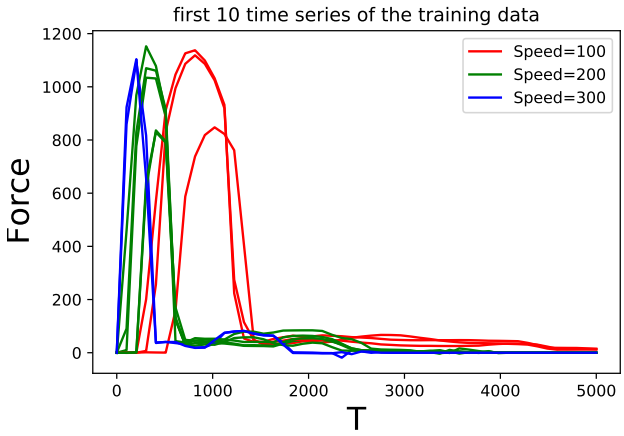


Abbildung 2: Visualisierung der Klassifikationsaufgabe.

1.1c) Visualisierung (1 Punkt)

Geben Sie die erstelle Visualisierung des Entscheidungsbaumes zur Klassifikation aus Unteraufgabe a) an.

Lösungsvorschlag:

s. Abb. 3.

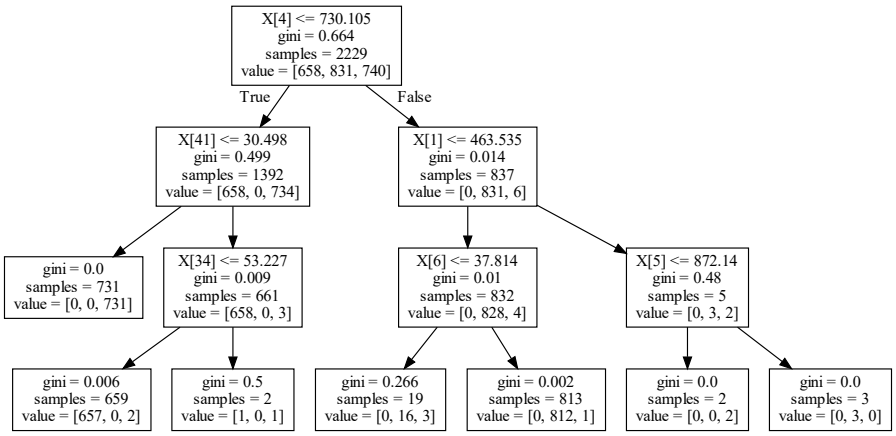


Abbildung 3: Struktur des Entscheidungsbaumes zur Klassifikation.

1.1d) Konfusionsmatrix (1 Punkt)

Geben Sie die Konfusionsmatrizen des Trainings- und Testdatensatzes mit absoluten Werten für den AdaBoost-Klassifier an.

Lösungsvorschlag:

training	100	200	300
100	658	0	0
200	0	831	0
300	0	0	740

test	100	200	300
100	166	0	0
200	0	207	0
300	0	4	181

Aufgabe 1.2: Entscheidungsbäume - Regression (10)

Ein Entscheidungsbaum zur Regression kann auf den PtU Datensatz (s. Abb. 1) angewendet werden, um auf die Dicke des Materials zu schließen, welche ein kontinuierlicher Wert ist.

1.2a) `</>` `02_ptu_regression.py` (7 Punkte)

Laden Sie die Daten des Kraftsensor aus `Filename_Fz_raw.csv` und die Materialstärken aus `Filename_thickness.csv` und vervollständigen Sie den Code in `02_ptu_regression.py`:

- `</>` Visualisieren Sie die ersten zehn Zeitreihen des Trainingsdatensatzes als Linien-Plot in der Methode `visualize_thickness_data()`. Bilden Sie dabei die Werte der Dicke farblich auf eine Farbtabelle/Colormap ab und geben Sie eine Beschriftung der Colormap an. Vergessen Sie nicht die Achsen zu beschriften. (3 Punkte)
- `</>` Visualisieren Sie die Verteilung der Werte für die Dicke mittels eines Histogramms mit 50 Bins. Färben Sie die Bins farblich mittels der gleichen Farbtabelle ein wie in der Methode `visualize_thickness_data()`. Vergessen Sie nicht die Achsen zu beschriften. (2 Punkte)
- `</>` Trainieren Sie einen Entscheidungsbaum zur Regression in `fit_dt_regressor()`. (1 Punkt)
- `</>` Berechnen Sie den mittleren quadratischen Fehler für den Testdatensatz in der Funktion `get_test_mse()`. (1 Punkt)

```python
from sklearn import tree
from sklearn.model_selection import train_test_split
import numpy as np
import graphviz
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from util import get_x_down_sampled
import pydotplus

def fit_dt_regressor(X_train: np.ndarray, y_train: np.ndarray, max_depth: int = None) -> tree.
    DecisionTreeRegressor:
    """Creates and fits a regression tree on the training data."""

    # max_depth:The maximum depth of the tree
    # Use sklearn.tree.DecisionTreeRegressor()
    regressor = tree.DecisionTreeRegressor(max_depth=max_depth)

    # fit():Build a decision tree regressor from the training set(X_train, y_train)
    regressor.fit(X_train,y_train)

    return regressor


def get_test_mse(clf, X_test: np.ndarray, y_test: np.ndarray) -> float:
    """Evaluates the test mse for a given classifier and test dataset."""

    y_test_pred = clf.predict(X_test)
    # Return mean squared error regression loss
    return mean_squared_error(y_test,y_test_pred)


def export_tree_plot(clf, filename: str) -> None:
    """Exports the tree plot for the given classifier as a pdf with given filename."""

    # Export the decision tree regressor process as a graphviz.dot file.
    dot_data = tree.export_graphviz(clf)

```

```
    # A Dot class will be returned to represent the graph
    graph = pydotplus.graph_from_dot_data(dot_data)

    # Generate PDF file
    graph.write_pdf(filename)


def visualize_thickness_data(X_train: np.ndarray, y_train: np.ndarray, nb_samples_to_plot: int) -> None:
    """Visualize the data by color encoding the thickness variable on a colormap and export the plot as
    pdf."""

    # print(X_train.shape)(2229,50)
    # print(y_train.shape)(2229,)

    # nb_samples_to_plot=10
    X_train_plot = X_train[:nb_samples_to_plot]
    y_train_plot = y_train[:nb_samples_to_plot]

    plt.figure()
    t = np.linspace(0, 5000, X_train.shape[1])
    # Xaxis:t, Yaxis:=X_train[:10], true_label:y_train[:10]
    for Y,true_label in zip(X_train_plot,y_train_plot):
        plt.plot(t,Y,c=plt.cm.inferno((true_label-np.min(y_train_plot))/(np.max(y_train_plot)-np.min(
    y_train_plot))))

    sm=plt.cm.ScalarMappable(cmap="inferno",norm=plt.Normalize(np.min(y_train_plot),np.max(y_train_plot)))
    cb=plt.colorbar(sm)
    cb.set_label("Thickness",fontsize=20)
    plt.xlabel("T",fontsize=20)
    plt.ylabel("Force",fontsize=20)
    plt.savefig('./thickness_data.pdf')
    plt.show()


def plot_thickness_histogram(y_train: np.ndarray, bins: 50) -> None:
    """Plots a histogram of 50 bins of the thickness distribution."""

    plt.figure()
    N,Bins,Patches=plt.hist(y_train,bins)
    for i in range(bins):
        Patches[i].set_facecolor(plt.cm.inferno(((Bins[i]+Bins[i+1])/2-np.min(y_train))/(np.max(y_train)-
    np.min(y_train))))
    plt.xlabel("Thickness",fontsize=20)
    plt.ylabel("Frequency",fontsize=20)
    sm=plt.cm.ScalarMappable(cmap="inferno",norm=plt.Normalize(np.min(y_train),np.max(y_train)))
    cb=plt.colorbar(sm)
    cb.set_label("Thickness",fontsize=20)
    plt.savefig('./thickness_histogram.pdf')
    plt.show()

def main():
    # for reproducibility
    np.random.seed(42)

    # load data
    X_data = get_x_down_sampled('./PtU/FileName_Fz_raw.csv')
    y_data = np.loadtxt(open('./PtU/FileName_thickness.csv', 'r'), delimiter=",", skiprows=0)
    print("X_data.shape:", X_data.shape)
    print("y_data.shape:", y_data.shape)

    # split training and test sets
    X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.2, random_state=42)

    # visualize the task
    visualize_thickness_data(X_train, y_train, 10)
    plot_thickness_histogram(y_train, 50)

    # train
    clf = fit_dt_regressor(X_train, y_train)
    # predict % evaluate
```

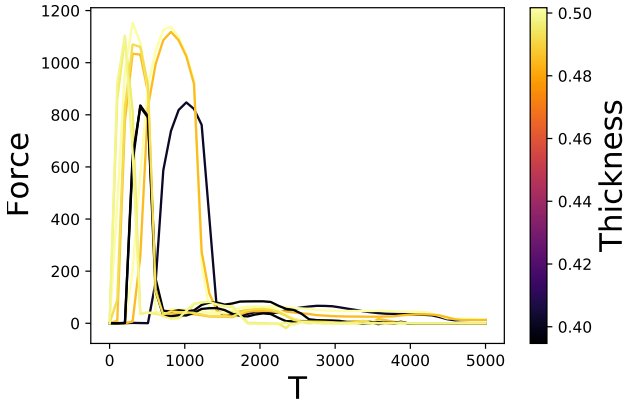
```
    mse = get_test_mse(clf, X_test, y_test)
    print('Test MSE:', mse)

    # change max tree depth
    # train
    clf = fit_dt_regressor(X_train, y_train, max_depth=3)

    # predict & evaluate
    mse = get_test_mse(clf, X_test, y_test)
    print('Test MSE:', mse)

    # plot tree
    export_tree_plot(clf, "regression_tree_d3.pdf")


if __name__ == '__main__':
    main()
```

1.2b) Datenvisualisierung (2 Punkte)

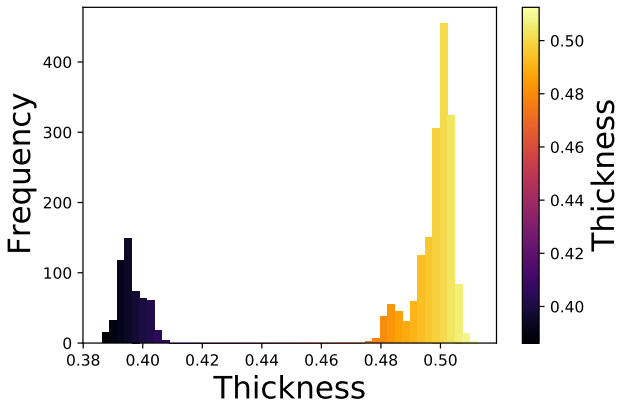
Geben Sie die erstellte Visualsierung der Daten und das Histogramm aus Unteraufgabe a) an.

Lösungsvorschlag:

s. Abb. 4.



(a) Materialdicke der ersten zehn Zeitreihen des Trainingsdatensatzes.



(b) Verteilung der Materialdicke des Trainingsdatensatzes.

Abbildung 4: Visualisierung der Regressionsaufgabe.

1.2c) Visualisierung (1 Punkt)

Geben Sie die erstellen Visualisierung des Entscheidungsbaumes zur Regression aus Unteraufgabe a).

Lösungsvorschlag:

s. Abb. 5.

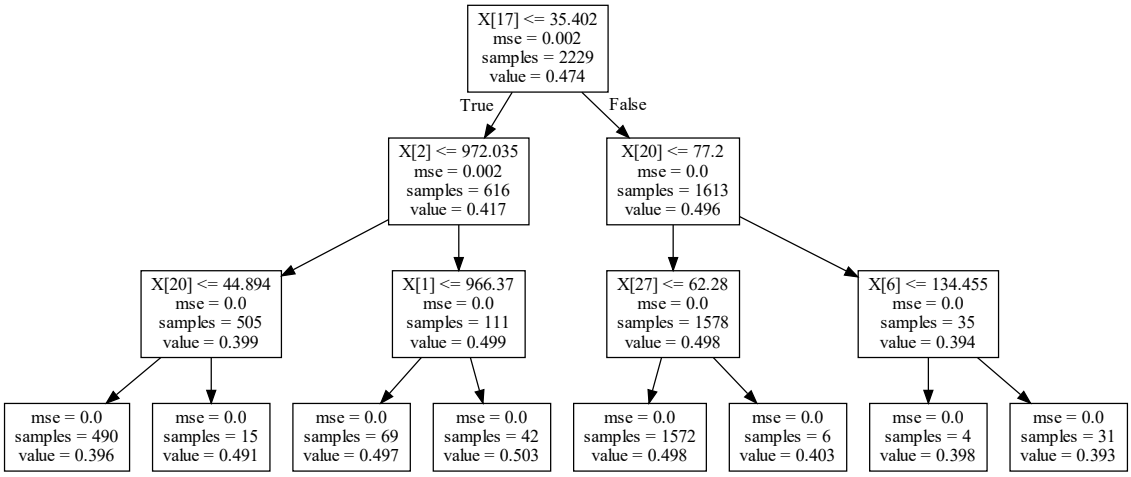


Abbildung 5: Struktur des Entscheidungsbaumes zur Regression.

Aufgabe 1.3: </> Dimensionsreduktion und Klassifikation (18)

In dieser und den darauf folgenden Aufgaben werden wir den `10 Monkey Species` Datensatz (s. Abb. 6) verwenden.

Laden Sie den Datensatz auf auf `kaggle.com` herunter.
`https://www.kaggle.com/slothkong/10-monkey-species`

Das Ziel des Datensatzes ist es die Bilder von zehn verschiedenen Affenarten korrekt mittels Verfahren des maschinellen Lernens richtig zu klassifizieren.

(a) Klasse n0: Mantelbrüllaffe (alouatta palliata)

(b) Klasse n7: Gewöhnlicher Totenkopfaffe (saimiri sciureus)

Abbildung 6: Beispielphoto der Klasse *saimiri sciureus* aus dem Datensatz `10 Monkey Species`

1.3a) `03_monkey_traditional_ml.py` (16 Punkte)

Vervollständigen Sie den Code in `03_monkey_traditional_ml.py`:

</> Implementieren Sie die Methode `get_train_val_data()`, welche den Trainings- und Validierungsdatensatz aus dem Daten-Verzeichnis einliest und auf eine vorgegebene Bildgröße von 224×224 skaliert. Zum Laden und

Skalieren der Bilder können Sie die OpenCV Bibliothek verwenden. Wenn Sie nicht den vollständigen Datensatz in den Arbeitsspeicher laden können, können Sie eine kleinere Auflösung verwenden. Die Methode soll den Trainings- und Validierungsdatensatz als numpy-Array mit den drei Farbkanälen RGB zurückgeben. Die zehn Klassen sollen als Integer-Werte $\in [0, 9]$ beschrieben werden. (3 Punkte)

</> Führen Sie in der Methode `apply_pca()` eine Dimensionalitätsreduktion auf zwei Dimensionen durch und geben Sie den transformierten Trainings- und Validierungsdatensatz zurück. Beachten Sie, dass Sie die Eigenwerte und Eigenvektoren nur anhand des Trainings- und nicht anhand des Validierungsdatensatzes bestimmen. Die Methode `apply_pca()` soll Tensoren mit 4 Dimensionen (z.B. Bilddaten), als auch Tensoren mit 2 Dimensionen (z.B. Matrizen), verarbeiten können. (3 Punkte)

</> Visualisieren Sie den transformierten Trainingsdatensatz als 2D-Scatterplot in der Methode `plot_pca_data()`. Stellen Sie die unterschiedlichen Klassen farblich dar und beschriften Sie die Achsen. (1 Punkt)

</> Evaluieren Sie in `evaluate_knn()` mittels des dimensionsreduzierten Datensatzes die Performance eines k-Nächste-Nachbarn Klassifiers für k=1. (2 Punkte)

</> Extrahieren Sie ein Farbhistogramm mit 32 Bins pro Farbe für den Datensatz in der Methode `get_histogram_data()`. Pro Zeile sollen die Bins der Farbkanäle nacheinander angehängt werden. Also 96 Einträge pro Zeile, wobei die ersten 32 für Farbe Rot, die Einträge 33-64 für Farbe Grün und 65-96 für Farbe Blau reserviert sind. (3 Punkte)

</> Trainieren Sie einen Klassifier Ihrer Wahl in der Methode `train_custom_classifier()` mittels des gegeben Datensatzes. Sie können dabei einen beliebigen Modelltyp der `sklearn` Bibliothek unter folgenden Einschränkung verwenden: Die Trainingszeit sollte unter 5 Minuten auf einer herkömmlichen CPU betragen und Sie dürfen kein neuronales Netz / Multi-Layer-Perceptron verwenden. Versuchen Sie eine Validierungsgenauigkeit von $\geq 35\,\%$ zu erreichen. (4 Punkte)

Hinweis: Sie können Merkmale kombinieren und müssen nicht zwangsläufig alle Merkmale verwenden.

Lösungsvorschlag:

```python
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn import preprocessing


def get_train_val_data(data_dir: str, shape: tuple) -> (np.ndarray, np.ndarray, np.ndarray, np.ndarray):
    """Reads the training and validation image data from the given data directory and
     scales all images to the given to the input shape.
     The target vectors (y_train, y_val) should contain the class labels as integer values."""

    import glob
    train_folder = os.path.join(data_dir,"training/training")
    x = []
    y = []

    for i in range(10):
        # for filename in glob.glob(train_folder + "/n{}/*.jpg".format(i)):
        for filename in glob.glob(train_folder + "/n{}/*".format(i)):
            image = cv2.imread(filename)
            image = cv2.resize(image, shape) # resize the image
            image_RGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # change to RGB
            x.append(image_RGB)
            y.append(i)
    X_train = np.array(x)
    y_train = np.array(y)

    # print("Training Dataset:")
    # print(X_train.shape)
    # print(y_train.shape)

    test_folder = os.path.join(data_dir,"validation/validation")
```

```
    x = []
    y = []

    for i in range(10):
        for filename in glob.glob(test_folder + "/n{}/*".format(i)):
            image = cv2.imread(filename)
            image = cv2.resize(image, shape) # resize the image
            image_RGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # change to RGB
            x.append(image_RGB)
            y.append(i)
    X_val = np.array(x)
    y_val = np.array(y)


    # print("Validation Dataset:")
    # print(X_val.shape)
    # print(y_val.shape)
    return X_train, y_train, X_val, y_val

def plot_pca_data(X_data_pca: np.ndarray, y_data: np.ndarray, filename: str):
    """Plots the first two principal components in a 2d scatter plot and saves the figure."""
    x1 = X_data_pca[:, 0]
    x2 = X_data_pca[:, 1]
    y = y_data

    plt.scatter(x1, x2, c=y)
    plt.title(filename)
    plt.xlabel('the first principal component in X_data_pca')
    plt.ylabel('the second principal component in X_data_pca')
    plt.show()



def get_histogram_data(X_data, nb_bins, color_channels=3) -> np.ndarray:
    """Extracts a histogram for each channel and returns the features """
    x=[]

    for i in range(X_data.shape[0]):
        img = X_data[i,:,:,:]
        hist_r = cv2.calcHist([img], [0], None, [nb_bins], [0, 255])
        hist_g = cv2.calcHist([img], [1], None, [nb_bins], [0, 255])
        hist_b = cv2.calcHist([img], [2], None, [nb_bins], [0, 255])
        hist_end = (np.concatenate([hist_r, hist_g, hist_b])).T
        x.append(hist_end)
    X_hist_basis = np.array(x)
    X_hist = X_hist_basis.squeeze()
    return X_hist



def evaluate_knn(X_train_pca, y_train, X_val_pca, y_val, k: int) -> (float, float):
    """Evaluates a k-nearest-Neighbour classifier with value k and returns the training and validation
    accuracy."""
    classifier = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p=2)
    classifier.fit(X_train_pca, y_train)
    train_acc = classifier.score(X_train_pca, y_train)
    val_acc = classifier.score(X_val_pca, y_val)

    return  train_acc, val_acc



def apply_pca(X_train: np.ndarray, X_val: np.ndarray, n_components: int) -> (np.ndarray, np.ndarray):
    """Returns the transformed data for X_train and X_val with n_components as principal components."""
    # if len(X_train.shape)>2:
    #     X_train = (X_train.reshape(X_train.shape[0], X_train.shape[1] * X_train.shape[2] * X_train.shape
    [3]))
    #
    # if len(X_val.shape)>2:
    #     X_val = (X_val.reshape(X_val.shape[0], X_val.shape[1] * X_val.shape[2] * X_val.shape[3]))
```

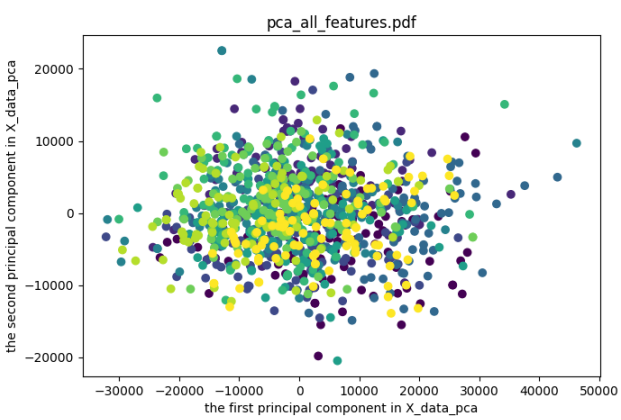
```
    X_train_reshape = X_train.reshape(X_train.shape[0],-1)
    X_train = X_train_reshape - np.mean(X_train_reshape, axis= 0).reshape(1,-1)
    X_val_reshape = X_val.reshape(X_val.shape[0],-1)
    X_val = X_val_reshape - np.mean(X_val_reshape,axis = 0).reshape(1,-1)



    pca = PCA(n_components=n_components)
    pca.fit(X_train)
    X_train_pca = pca.transform(X_train)
    X_val_pca = pca.transform(X_val)
    return X_train_pca, X_val_pca


def train_custom_classifier(X_train, X_train_pca, X_train_pca_hist,
                            X_val, X_val_pca, X_val_pca_hist,
                            y_train, y_val) -> (float, float):
    """Trains a custom non neural network classifier and returns the training and validation accuracy."""


    X_train_forest = np.hstack((X_train_pca, X_train_pca_hist))
    rfc = RandomForestClassifier(random_state=42)
    rfc.fit(X_train_forest, y_train)
    train_acc = rfc.score(X_train_forest, y_train)
    X_val_forest = np.hstack((X_val_pca, X_val_pca_hist))
    val_acc = rfc.score(X_val_forest, y_val)
    return train_acc, val_acc

def main():
    # for reproducibility
    np.random.seed(42)

    data_dir = './data/kaggle/10-monkey-species'
    print('Reading image data...')
    get_train_val_data(data_dir, (224, 224))
    X_train, y_train, X_val, y_val = get_train_val_data(data_dir, (224, 224))

    print('Apply PCA on original data...')
    X_train_pca, X_val_pca = apply_pca(X_train, X_val, 2)
    plot_pca_data(X_train_pca, y_train, 'pca_all_features.pdf')
    train_acc, val_acc = evaluate_knn(X_train_pca, y_train, X_val_pca, y_val, 1)
    print(f"knn (k=1) Validation-Acc: {val_acc}")

    X_train_hist = get_histogram_data(X_train, 32)

    X_val_hist = get_histogram_data(X_val, 32)
    print('Apply PCA on color histogram data...')
    X_train_pca_hist, X_val_pca_hist = apply_pca(X_train_hist, X_val_hist, 2)
    plot_pca_data(X_train_pca_hist, y_train, 'pca_color_histogram.pdf')
    train_acc, val_acc = evaluate_knn(X_train_pca_hist, y_train, X_val_pca_hist, y_val, 1)
    print(f"knn (k=1) Validation-Acc: {val_acc}")

    train_acc, val_acc = train_custom_classifier(X_train, X_train_pca, X_train_pca_hist,
                                                 X_val, X_val_pca, X_val_pca_hist,
                                                 y_train, y_val)

    print(f"Validation Acc. for custom classifier: {val_acc}")


if __name__ == '__main__':
    main()
```

1.3b) Datenvisualisierung (2 Punkte)

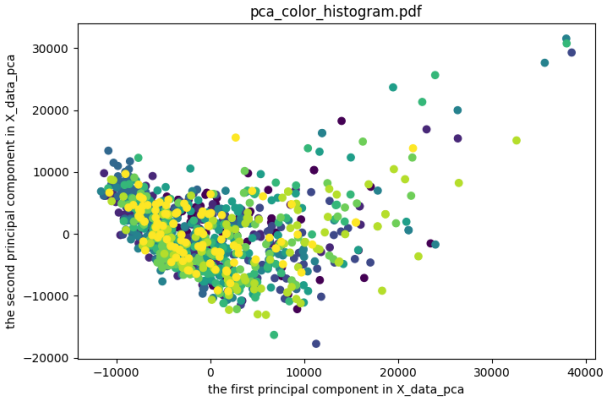
Geben Sie die beiden Scatterplots der Datenverteilung nach Transformation mittels PCA aus Unteraufgabe a) an.

Lösungsvorschlag:

s. Abb. 7.



(a) Datenverteilung nach Anwendung von PCA auf Rohdaten.



(b) Datenverteilung nach Anwendung von PCA auf Farbhistogramme.

Abbildung 7: Datenverteilung nach Transformation mittels PCA.

Aufgabe 1.4: </> Einfaches Convolutional Neural Network (12)

In dieser Aufgabe verwenden wir den `10 Monkey Species` Datensatz (s. Abb. 6), um ein einfaches Convolutional Neural Networks (CNN) zu trainieren.

Verwenden Sie ein Deep Learning Framework Ihrer Wahl und eine passende Trainingskonfiguration. Wenn Sie mit noch keinen Deep Learning Framework vertraut sind, können wir Keras als Einstiegsframework empfehlen.

Wenn Sie Ihre Kenntnisse in Deep Learning weiter vertiefen möchten können Sie den Kurs *Deep Learning Methods & Architectures* (20-00-1034-iv) im nächsten Sommersemester besuchen. Ergänzende Literatur zum Thema Deep Learning und Transfer Learning finden Sie in *Modern mathematics of Deep Learning* [1] und *The Principles of Deep Learning Theory* [2].

1.4a) </> `04_monkey_simple_cnn.ipynb` (10 Punkte)

Implementieren Sie Ihre Lösung im Jupyter-Notebook `04_monkey_simple_cnn.ipynb` und geben Sie das Jupyter-Notebook samt Trainingslog ab.

</> Erstellen und trainieren Sie ein einfaches Convolutional Neural Network mit bis zu vier Convolutional Layern. Sie können dabei Max-Pooling Layer oder Strided Convolution verwenden.

Beim Trainingsvorgang können Sie ebenfalls Datenaugmentation verwenden.

Versuchen Sie eine Validierungsgenauigkeit von $\geq 50\,\%$ zu erreichen. Die Trainingszeit sollte unter 15 Minuten auf einer NVIDIA GTX 1080ti vergleichbaren GPU betragen. (10 Punkte)

Hinweis: Sollte keiner ihrer Gruppenmitglieder eine GPU zur Verfügung haben, können Sie stattdessen mittels CPU trainieren oder Google Colab nutzen:
https://colab.research.google.com/notebooks/intro.ipynb

```python
import pandas as pd
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, initializers,preprocessing
import matplotlib.pyplot as plt
from PIL import Image
# todo
### Load the image data or define the data loader

train_path = '/Users/llm/Desktop/dmml_bonus_2021_scripts_2/archive/training/training'
valid_path = '/Users/llm/Desktop/dmml_bonus_2021_scripts_2/archive/validation/validation'
label_path = '/Users/llm/Desktop/dmml_bonus_2021_scripts_2/archive/monkey_labels.txt'

# data augmentation
'''
reference : https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/
    ImageDataGenerator

'''
train_datagen = preprocessing.image.ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 20,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True,
    fill_mode = "nearest"
)
train_generator = train_datagen.flow_from_directory(
    train_path,
    target_size = (128, 128),
    batch_size = 64,
    seed = 7,
    shuffle = True,
    class_mode = 'categorical'
)
valid_datagen = preprocessing.image.ImageDataGenerator(
```

```
    rescale = 1./255
)
valid_generator = valid_datagen.flow_from_directory(
    valid_path,
    target_size = (128, 128),
    batch_size = 64,
    seed = 7,
    shuffle = False,
    class_mode = 'categorical')
train_num = train_generator.samples
valid_num = valid_generator.samples


# ### Define your model architecture

'''
Refernce : https://www.tensorflow.org/tutorials/images/cnn?hl=zh-eg

'''
model = models.Sequential()
model.add(layers.Conv2D(32, kernel_size=3, activation='relu', input_shape=[128, 128, 3]))
model.add(layers.MaxPooling2D(pool_size=2))
model.add(layers.Conv2D(64, kernel_size=3, activation='relu'))
model.add(layers.MaxPooling2D(pool_size=2))
model.add(layers.Conv2D(64, kernel_size=3, activation='relu'))
model.add(layers.MaxPooling2D(pool_size=2))
model.add(layers.Conv2D(64, kernel_size=3, activation='relu'))
model.add(layers.MaxPooling2D(pool_size=2))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()

# ### Define your training setup

# todo
model.compile(optimizer='adam',
              loss="categorical_crossentropy",
              metrics=['accuracy'])


# ### Initialize your model weights
# todo
'''
Reference: https://keras.io/api/layers/initializers/

'''

initializers.TruncatedNormal(mean=0.0, stddev=0.05, seed=None)

# ### Run the training process

'''
Reference: https://www.tensorflow.org/tutorials/images/cnn?hl=zh-eg

'''

history = model.fit_generator(train_generator,
                              steps_per_epoch=train_num // 64,
                              epochs=20,
                              validation_data=valid_generator,
                              validation_steps=valid_num // 64)

# ### Plot the learning curves

'''
Reference: https://www.tensorflow.org/tutorials/images/cnn?hl=zh-eg

'''

```

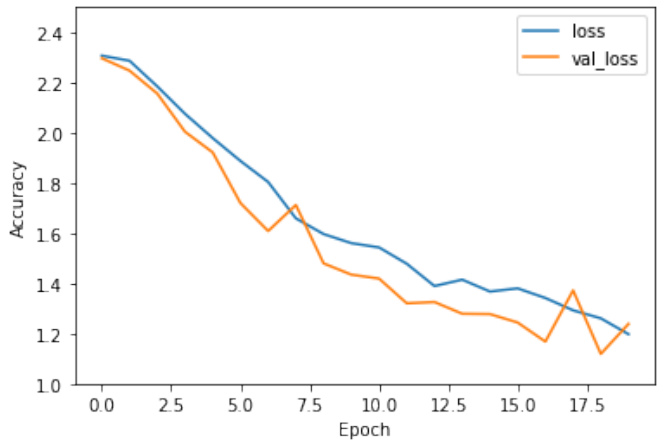
```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.0, 1])
plt.legend()
plt.show()
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([1.0, 2.5])
plt.legend()
plt.show()
```

1.4b) Lernplots (2 Punkte)

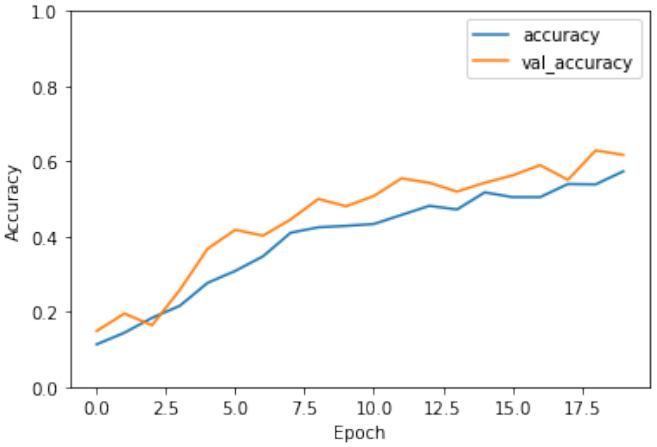
Geben Sie hier in zwei Plots den Verlauf des Trainings- und Validierungsfehlers sowie der Trainings- und Validierungsgenauigkeit an.

Lösungsvorschlag:

s. Abb. 8.



(a) Entwicklung des Trainings- und Validierungsfehlers.



(b) Entwicklung der Trainings- und Validierungsgenauigkeit.

Abbildung 8: Lernplots für ein einfaches Convolutional Neural Network.

Aufgabe 1.5: </> Convolutional Neural Network - Transfer Learning (18)

In dieser Aufgabe verwenden wir den `10 Monkey Species` Datensatz (s. Abb. 6), um Transfer Learning anzuwenden.

1.5a) </> `05_monkey_transfer_cnn.ipynb` (15 Punkte)

Implementieren Sie Ihre Lösung im Jupyter-Notebook `05_monkey_transfer_cnn.ipynb` und geben Sie das Jupyter-Notebook samt Trainingslog ab.

</> Verwenden Sie ein vortrainiertes Deep-Learning-Modell aus dem Modell-Zoo des gewählten Deep-Learning Frameworks und transformieren Sie damit die Bilddaten nach der Ausgabe der letzten Convolutional Layer. (4 Punkte)

Hinweis: Als Empfehlung können Sie **MobileNetV1** verwenden. Dieses ist auf

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

verfügbar und kann mittels der Keras API heruntergeladen werden.

</> Wenden Sie auf diesen durch das CNN transformierten Daten erneut eine Dimensionsreduktion nach PCA auf zwei Dimensionen an. (2 Punkte)

</> Visualisieren Sie analog zu Aufgabe 3a) diesen 2d-Datensatz in einem Scatterplot. (1 Punkt)

</> Evaluieren Sie analog zu Aufgabe 3a) diesen 2d-Datensatz mittels eines k-Nächste-Nachbarn Klassifiers, um die Affenarten zu klassifizieren. (2 Punkte)

</> Verwenden Sie das gleiche vortrainierte Modell aus dem Modell-Zoo des gewählten Deep-Learning Frameworks und fügen Sie ein passenden Klassifizierungskopf an. Optimieren Sie die Parameter der neu angefügten Schichten des neuronalen Netzwerkes. Die Parameter der vorangehenden Schichten bleiben fixiert. Beim Trainingsvorgang können Sie ebenfalls Datenaugmentation verwenden.

Geben Sie die erreichte Trainings- und Validierungsgenauigkeit an. Die Trainingszeit sollte unter 15 Minuten auf einer NVIDIA GTX 1080ti vergleichbaren GPU betragen. Finden Sie eine Konfiguration um eine Validierungsgenauigkeit $\geq 95\,\%$ zu erreichen. (6 Punkte)

```
# In[137]:


import tensorflow as tf
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from sklearn.decomposition import PCA
from matplotlib.colors import ListedColormap
from sklearn.neighbors import KNeighborsClassifier
from keras.callbacks import ModelCheckpoint


# ### Load the image data or define the data loader

# In[138]:


cols = ['Label', 'Latin Name', 'Common Name', 'Train Image', 'Validation Images']
monkeys = pd.read_csv(f"monkey_labels.txt", names=cols, skiprows=1)
height = 224
width = 224
#make the data generator.
train_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    fill_mode='nearest')
train_generator = train_datagen.flow_from_directory(
    'training/training/',
    target_size=(height, width),
```

```
    batch_size=1,
    shuffle=False)
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = train_datagen.flow_from_directory(
    'validation/validation/',
    target_size=(height, width),
    batch_size=1,
    shuffle=False)


# In[139]:


monkeys


# ### Load a pre-trained model

# In[140]:


mobileNet = tf.keras.applications.MobileNet(input_shape=(224, 224, 3), include_top=False, weights='
    imagenet')
mobileNet.trainable = False


# ### Transform the image data to the features of the last convolutional layer

# In[142]:


features = np.zeros([1098, 7, 7, 1024])
labels = np.zeros(1098)
labels_oh = np.zeros([1098, 10])
for i in range(1098):
    feature = mobileNet.predict(train_generator[i][0])
    features[i] = feature
    labels_oh[i] = train_generator[i][1].reshape(-1)
    labels[i] = train_generator[i][1].reshape(-1).argmax()
    if i % 100 == 0:
        print("finished {} procent!".format(i // 10))


# ### Apply PCA to the transformed CNN features

# In[143]:


features = features.reshape([1098, 7*7*1024])
pca_model = PCA(n_components=2)
features_d = pca_model.fit_transform(features)


# ### Plot the two dimensional data distribution

# In[145]:


cmap = ListedColormap(['b', 'r', 'g', 'y', 'c', 'm', 'k', 'olive', 'pink', 'purple', 'orange'])
plt.figure()
plt.scatter(features_d[:, 0], features_d[:, 1], c=labels, marker='.', cmap=cmap, edgecolors='none',
    label="train data")
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.savefig("two_dim_data")
plt.show()


# ### Evaluate a kNN with k=1 on the two dimensional data distribution

```

```
# In[146]:


cls = KNeighborsClassifier(n_neighbors=1)
cls.fit(features_d, labels)
features_test = np.zeros([272, 7, 7, 1024])
y_label = np.zeros([272])
label_test_oh = np.zeros([272, 10])
for i in range(272):
    features_test[i] = mobileNet(test_generator[i][0])
    y_label[i] = test_generator[i][1].reshape(-1).argmax()
    label_test_oh[i] = test_generator[i][1].reshape(-1)
features_test_d = pca_model.transform(features_test.reshape([272, 7*7*1024]))
y_pred = cls.predict(features_test_d)


# In[147]:


acc = (y_pred == y_label).sum() / 272
print("The accuracy of validation sets is: {}".format(acc))


### Add a custom classification head on the pre-trained model

# In[148]:


fc_model = tf.keras.Sequential([
    tf.keras.layers.Dense(5000, activation='relu'),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
fc_model.build(input_shape=[None, 7*7*1024])
fc_model.summary()


### Define your training setup

# In[149]:


epochs = 50
batch_size = 64
checkpoint = ModelCheckpoint("monkey_model.h5f", monitor='val_acc', verbose=1, save_best_only=True,
    mode='max')
ls=tf.keras.losses.CategoricalCrossentropy()
features_test = features_test.reshape([272, 7*7*1024])
fc_model.compile(
    optimizer = 'sgd',
    loss = tf.keras.losses.CategoricalCrossentropy(from_logits = False),
    metrics = ['acc'])


### Run the training process

# In[150]:


history = fc_model.fit(
    features, labels_oh,
    batch_size = batch_size,
    validation_data = (features_test, label_test_oh),
    steps_per_epoch = 5,
    epochs = epochs,
    callbacks=[checkpoint],
    verbose=1)


### Plot the learning curves
```

```
169
170 # In[152]:
171
172
173 plt.figure()
174 steps = np.arange(1, epochs)
175 plt.plot(steps, history.history["loss"][1:], label="training sets")
176 plt.plot(steps, history.history["val_loss"][:-1], label="validation sets")
177 plt.xlabel("step")
178 plt.ylabel("loss")
179 plt.title("loss of training sets and test sets")
180 plt.legend()
181 plt.savefig("loss_pic.png")
182 plt.show()
183 plt.figure()
184 plt.plot(steps, history.history["acc"][1:], label="training sets")
185 plt.plot(steps, history.history["val_acc"][1:], label="validation sets")
186 plt.xlabel("step")
187 plt.ylabel("accuracy")
188 plt.title("accuracy of training sets and test sets")
189 plt.legend()
190 plt.savefig("acc_pic.png")
191 plt.show()
192
193
194 # Here we use this model to predict two monkeys. From the training step, we can see that the accuracy
        of the most accurate model is 0.9596. This model is good to estimate the class of monkey.
195
196 # In[153]:
197
198
199 def predict_monkey(mobileNet, fc_model, testdata):
200     fc_model.trainable = False
201     real_name = monkeys["Common Name"][int(tf.argmax(testdata[1][0]))]
202     features = tf.reshape(mobileNet.predict(testdata[0]), [-1, 50176])
203     result = int(tf.argmax(tf.reshape(fc_model.predict(features), [10])))
204     pred_name = monkeys["Common Name"][result]
205     plt.figure()
206     plt.imshow(testdata[0][0])
207     plt.title("Predicted label is: " + real_name + "\nreal label is: " + pred_name)
208 predict_monkey(mobileNet, fc_model, test_generator[50])
209 predict_monkey(mobileNet, fc_model, test_generator[100])
```

1.5b) Datenvisualisierung: PCA (1 Punkt)

Geben Sie hier den Scatterplot der Datenverteilung nach Transformation mittels PCA aus Unteraufgabe a) an.

Lösungsvorschlag:

s. Abb. 9.

1.5c) Lernplots (2 Punkte)

Geben Sie hier in zwei Plots den Verlauf des Trainings- und Validierungsfehlers sowie der Trainings- und Validierungsgenauigkeit an.

Lösungsvorschlag:

s. Abb. 10.

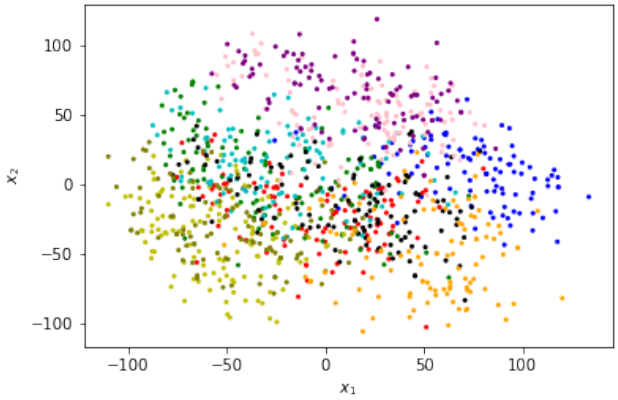
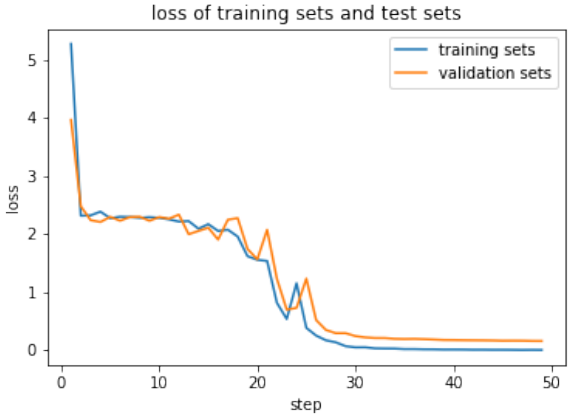
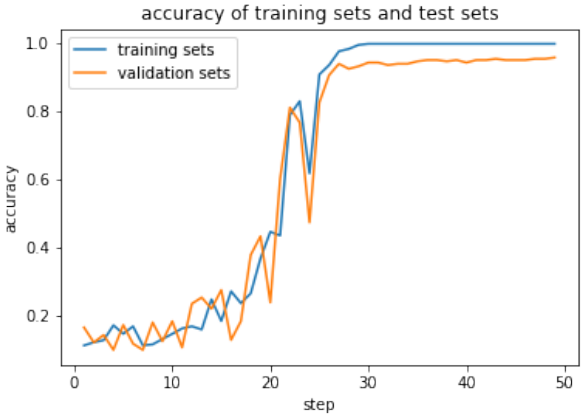


Abbildung 9: Datenverteilung der Merkmale des vortrainierten CNNs nach Transformation mittels PCA.



(a) Entwicklung des Trainings- und Validierungsfehlers



(b) Entwicklung der Trainings- und Validierungsgenauigkeit

Abbildung 10: Übersicht der Lernplots des Trainingsvorgangs

Literatur

[1] Julius Berner, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen. The modern mathematics of deep learning. *arXiv preprint arXiv:2105.04026*, 2021.

[2] Daniel A Roberts, Sho Yaida, and Boris Hanin. The Principles of Deep Learning Theory. *https://deeplearningtheory.com/PDLT.pdf*, page 449, 2021.