06_TransferLearning_Cheetahs

April 15, 2018

1 Übung 6: Transfer learning

Neuronal Netze benötigen oft eine groSSe Menge an Trainingsdaten, damit es nicht zu overfitting kommt. Transfer learning erlaubt es, mit relativ geringen Datenmenge dennoch erfolgreiche groSSe Netze zu trainieren. Dabei verwendet man ein bereits auf einen anderen Datensatz (z.b. ImageNet) vortrainiertes Netzwerk, und ersetzt nur das letzte Layer durch ein neues. In dieser Übung geht es darum, ein Netzwerk für die Erkennung von Geparden und Leoparden in der freien Wildbahn zu trainineren.

1.1 Daten laden

Lade die Daten hier herunter: http://tonic.imp.fu-berlin.de/cv_data/data.tar.gz

Die Daten wurde bereits in ein Trainings- und Validierungsset geteilt. Die Ordnerstruktur ist wie bei vielen Bildklassifierungsdatensetzen so aufgebaut. Es gibt zwei Unterordner für die Trainings- und Validierunsdaten. In diesen Ordnern liegen dann jeweils alle Bilder von einer Klasse in einem Unterordner mit dem Namen der Klasse.

Ein Beispiel: Die Trainingsbilder für die Klasse “cheetah” liegen in dem Unterordner train/cheetah

Diese Orderstruktur wird auch von dem in keras enhaltenen ImageDataGenerator unterstützt.

```
In [1]: import tensorflow as tf
        import itertools
        from keras.backend.tensorflow_backend import set_session, get_session
        import matplotlib.pyplot as plt
        from sklearn.metrics import confusion_matrix, roc_auc_score
        from sklearn.preprocessing import binarize
        import numpy as np
        import warnings; warnings.simplefilter('ignore')

        %matplotlib inline

        config = tf.ConfigProto()
        config.gpu_options.allow_growth = True
        set_session(tf.Session(config=config))

        import os
        from keras.preprocessing.image import ImageDataGenerator
```

```
from keras.applications.vgg16 import VGG16, decode_predictions

from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.pooling import MaxPooling2D
from keras.utils.np_utils import to_categorical
from keras import optimizers


def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion matrix',  cma

    # http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matr

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


def plots(ims, figsize=(12,6), rows=1, interp=False, titles=None):

    # https://github.com/deeplizard/Keras_Jupyter_Notebooks/blob/master/CNN.ipynb

    if type(ims[0]) is np.ndarray:
```

```
            ims = np.array(ims).astype(np.uint8)
            if (ims.shape[-1] != 3):
                ims = ims.transpose((0,2,3,1))
        f = plt.figure(figsize=figsize)
        cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
        for i in range(len(ims)):
            sp = f.add_subplot(rows, cols, i+1)
            sp.axis('off')
            if titles is not None:
                sp.set_title(titles[i], fontsize=16)
            plt.imshow(ims[i], interpolation=None if interp else 'none')
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/h5py/__init__.py
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

```
In [2]: batch_size = 32
        image_input_size = (112, 112)

        data_path = 'data/'
```

```
In [3]: train_path = os.path.join(data_path, 'train')
        valid_path = os.path.join(data_path, 'val')

        classes = ('unknown', 'cheetah', 'leopard')

        train_batches = ImageDataGenerator(horizontal_flip=True).flow_from_directory(
            train_path,
            target_size = image_input_size,
            classes = classes,
            batch_size = batch_size)

        valid_batches = ImageDataGenerator(horizontal_flip=False).flow_from_directory(
            valid_path,
            target_size = image_input_size,
            classes=classes,
            batch_size = batch_size,
            shuffle=False)
```

```
Found 17857 images belonging to 3 classes.
Found 1915 images belonging to 3 classes.
```

1.2 Training ohne transfer learning

Trainiere zuerst ein kleines Classifer-Netzwerk ohne transfer learning. Falls du keine Grafikkarte hast, solltest du nicht die volle Auflösung (siehe Variable image_input_size) verwenden, da das Training sonst zu lange dauert. Eine BildgröSSe von 32x32 Pixeln wäre zum Beispiel möglich.

In [4]:
```
input_shape = (image_input_size[0], image_input_size[1], 3)

lr = 0.001

model = Sequential()

model.add(MaxPooling2D(pool_size=(2, 2), data_format="channels_first", input_shape=inpu
model.add(Conv2D(256, (3, 3), strides=2, activation="relu"))
model.add(Dropout(0.25))
model.add(Conv2D(256, (3, 3), activation="relu"))
model.add(Conv2D(128, (3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2), data_format="channels_first"))

model.add(Conv2D(96, (3, 3), activation="relu"))
model.add(Conv2D(64, (3, 3), activation="relu"))
model.add(Dropout(0.25))
model.add(Conv2D(32, (3, 3), activation="relu"))
model.add(Flatten())

model.add(Dense(64, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.25))
model.add(Dense(32, activation="relu"))
model.add(Dense(3, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.Adam(lr=lr),
              metrics=['accuracy'])

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
max_pooling2d_1 (MaxPooling2 (None, 112, 56, 1)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 55, 27, 256)       2560
_________________________________________________________________
dropout_1 (Dropout)          (None, 55, 27, 256)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 53, 25, 256)       590080
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 51, 23, 128)       295040
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 51, 11, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 49, 9, 96)         55392
_________________________________________________________________
```

```
conv2d_5 (Conv2D)            (None, 47, 7, 64)         55360
_________________________________________________________________
dropout_2 (Dropout)          (None, 47, 7, 64)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 45, 5, 32)         18464
_________________________________________________________________
flatten_1 (Flatten)          (None, 7200)              0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                460864
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 99
=================================================================
Total params: 1,484,099
Trainable params: 1,484,099
Non-trainable params: 0
_________________________________________________________________
```

```
In [ ]: model.fit_generator(
                train_batches,
                steps_per_epoch = 1800 // batch_size,
                epochs = 10,
                validation_data = valid_batches,
                validation_steps = 250 // batch_size)

        model.load_weights('models/weights-8449.h5')
```

Erstelle eine Confusion matrix basierend auf den Ausgaben des Netzes für die Validierungsdaten und berechne den ROC AUC für die Klasse cheetah. Du kannst hierfür optional die scikit-learn Bibliothek verwenden.

```
In [ ]: predictions = model.predict_generator(valid_batches)

        valid_labels = valid_batches.classes

        cm = confusion_matrix(valid_labels, predictions.argmax(axis = 1))

        plot_confusion_matrix(cm, classes, title="Confusion Matrix")

        valid_labels_auc = np.copy(valid_labels) #if cheetah, set true (1)
        np.place(valid_labels_auc, valid_labels_auc == 2, 0)
```

```
roc_score = roc_auc_score(valid_labels_auc, predictions[:,1])
print(roc_score)
```

1.3 Pretrained network

Lade nun ein auf Imagenet vortrainiertes Netzwerk und klassifiziere damit die Validierungsdaten. Eine Anleitung für keras findest du hier: https://keras.io/applications

Du kannst selber entscheiden, welche Netzwerkarchitektur du verwendest.

```
In [ ]: model = ResNet50(weights = 'imagenet')
        model.summary()
```

Da der ImageNet-Datensatz auch die Klassen cheetah und leopard enthält, können wir sogar ohne transfer learning das vortrainierte Netzwerk evaluieren. Interpretiere alle Klassen auSSer cheetah und leopard als unknown und berechne wie im vorherigen Schritt die Confusion matrix und den ROC AUC score für die Klasse cheetah.

```
In [ ]: model_predictions = model.predict_generator(valid_batches)

        valid_labels = valid_batches.classes
```

1.4 Transfer learning

Das vortrainierte Netzwerk kann nun mit unseren Daten weitertrainiert werden. Ersetze dafür das letzte Layer in dem Netzwerk mit einem Dense Layer mit 3 Ausgaben für unsere Klassen cheetah, leopard und unknown. Du kannst selbst entscheiden, ob du nun das komplette Netzwerk mit trainierst oder nur das neu eingefügte, letzte Layer.

Auch hierfür kannst du dich wieder an der keras Anleitung orientieren: https://keras.io/applications

```
In [ ]: custom_model = Sequential()

        for layer in vgg16_model.layers:
            custom_layer = layer
            custom_layer.trainable = False
            custom_model.add(custom_layer)

        custom_model.layers.pop()
        custom_model.add(Dense(3, activation="softmax"))
        custom_model.summary()
```

Evaluiere das so trainierte Netzwerk wie in den letzten beiden Aufgaben.

```
In [ ]: # TODO
```

1.5 Auswertung

Beschreibe kurz qualitativ die Resultate. Wie unterscheiden sich die trainierten Netzwerke, zum Beispiel im Bezug auf die Genauigkeit oder die Laufzeit? Welche Entscheidungen musstest du bei der Erfüllung der Aufgaben treffen und warum hast du dich für den von dir gewählten Weg entschieden?