Natural Images

Zürcher Hochschule
für Angewandte Wissenschaften

Übersicht

Dieser Datensatz ("Natural Images") wurde von www.kaggle.com bezogen.

Anzahl Bilder	6899 + Augmentation: 20697
Klassenlabel	airplane (0), car (1), cat (2), dog (3), flower (4), fruit (5), motorbike (6), person (7)
Aufteilung	Jede Klasse besitzt 10-14% der Bilder. Alle Klassen besitzen somit ähnlich viele Bilder.

Bildvorbereitung

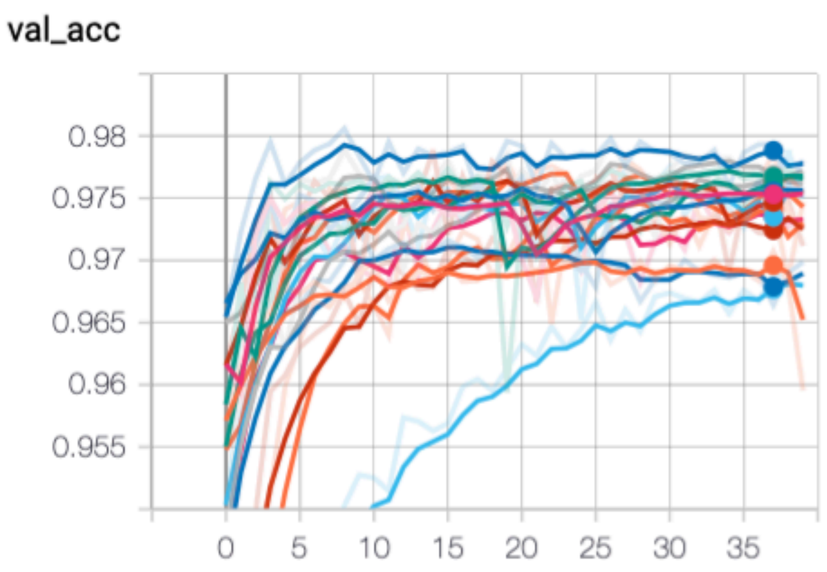
1. Datenanreicherung
   Jedes Bild wurde vertikal und horizontal gespiegelt und zu den Originalbildern abgespeichert.
2. Einlesen und Grössenänderung
   Alle Bilder werden in einem multidimensionalen Array eingelesen. Dabei wurden die Bilder auf eine Grösse von 50x50 Pixel skaliert.
3. Reihenfolge
   Randomisierte Neuorientierung der Datensätze im Array.
4. Aufteilung
   Trainingsdatensatz: 80% der Bilder
   Validierungsdatensatz: 30% der Trainingsdaten
   Testdatensatz: Restliche Bilder
5. Normalisierung
   Die Pixelwerte, sprich die Bilder, wurden normalisiert.
6. Binärdaten
   Die Labels wurden mittels ConvertToOneHot in Binärdaten umgewandelt.

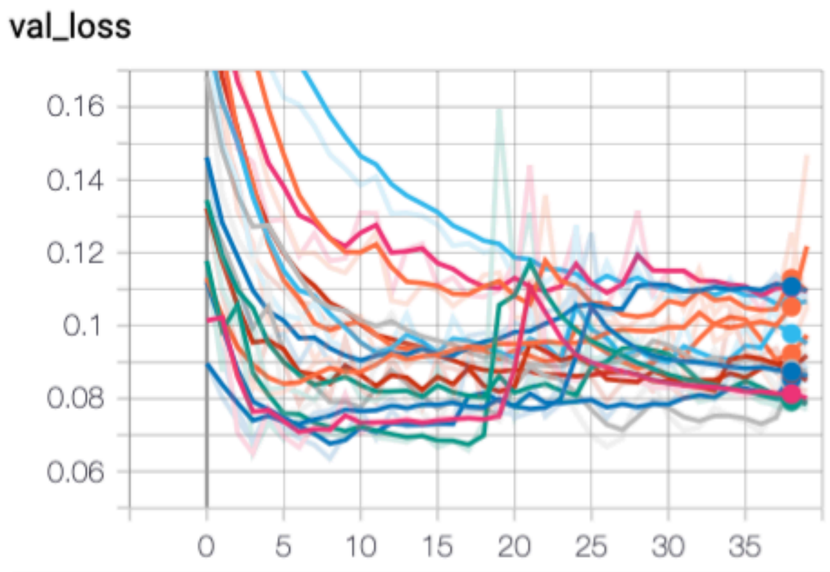
Natural Images

Zürcher Hochschule
für Angewandte Wissenschaften

Modellsuche

Die Leistungsfähigkeit eines neuronalen Netzes hängt von vielen Faktoren und Einstellungen ab - von der Anzahl und der Art der Schichten, die Grösse der Kerne, den Parametern der Fit-Funktion, die Grösse der Bilder etc. Nachdem wir ein paar Modell getestet hatten sind wir zu dem Entschluss gekommen (vor allem wegen der geringen Rechnerleistung, welche uns zur Verfügung stand) uns nur auf die Anzahl Schichten und den Nodes zu beschränken. In zwei For-Schleifen liessen wir 15 Modell trainieren und verwendeten dazu den kleineren Datensatz (6899 Bilder). Das resultierende beste Modell trainierten wir daraufhin mit dem grösseren Modell (20697 Bilder). Das Ergebnis lag aber nur geringfügig über dem Modells welches mit dem kleineren Datensatz trainiert wurde. Das beste Modell ergab sich mit 2 Schichten und 32 Notes ("2-conv-32-nodes-0-dense-...-aug").





Name	Smoothed	Value
1-conv-16-nodes-0-dense-1554278728	0.9753	0.9752
1-conv-32-nodes-0-dense-1554280026	0.9756	0.9758
1-conv-64-nodes-0-dense-1554282254	0.9767	0.9765
1-conv-8-nodes-0-dense-1554277769	0.9696	0.9707
2-conv-16-nodes-0-dense-1554278952	0.9762	0.9768
2-conv-32-nodes-0-dense-1554280365	0.9747	0.9752
2-conv-32-nodes-0-dense-1554290346-aug	0.9788	0.9793
2-conv-64-nodes-0-dense-1554282829	0.9769	0.9759
2-conv-8-nodes-0-dense-1554277944	0.9679	0.9664
3-conv-16-nodes-0-dense-1554279278	0.9768	0.9776
3-conv-32-nodes-0-dense-1554280929	0.9735	0.9717
3-conv-64-nodes-0-dense-1554284014	0.9764	0.9786
3-conv-8-nodes-0-dense-1554278161	0.9724	0.9721
4-conv-16-nodes-0-dense-1554279651	0.9741	0.9745
4-conv-32-nodes-0-dense-1554281579	0.9735	0.9733
4-conv-8-nodes-0-dense-1554278441	0.9677	0.9689

Name	Smoothed	Value
1-conv-16-nodes-0-dense-1554278728	0.08129	0.08095
1-conv-32-nodes-0-dense-1554280026	0.08736	0.08636
1-conv-64-nodes-0-dense-1554282254	0.07957	0.07894
1-conv-8-nodes-0-dense-1554277769	0.1052	0.1127
2-conv-16-nodes-0-dense-1554278952	0.07927	0.07677
2-conv-32-nodes-0-dense-1554280365	0.0871	0.08343
2-conv-32-nodes-0-dense-1554290346-aug	0.08549	0.09289
2-conv-64-nodes-0-dense-1554282829	0.08428	0.09849
2-conv-8-nodes-0-dense-1554277944	0.1108	0.1097
3-conv-16-nodes-0-dense-1554279278	0.08818	0.08931
3-conv-32-nodes-0-dense-1554280929	0.09781	0.09398
3-conv-64-nodes-0-dense-1554284014	0.09232	0.0948
3-conv-8-nodes-0-dense-1554278161	0.0883	0.08505
4-conv-16-nodes-0-dense-1554279651	0.1129	0.1254
4-conv-32-nodes-0-dense-1554281579	0.1105	0.1109
4-conv-8-nodes-0-dense-1554278441	0.1059	0.1057

Natural Images

Zürcher Hochschule
für Angewandte Wissenschaften

Modellsuche

Mit zwei For-Schleifen wurde systematisch mit verschiedenen Layer- und Nodes-Kombinationen trainiert. Eine weitere For-Schleife für Dense-Layer könnte noch hinzugefügt werden, wurde aber aufgrund der benötigten Rechenzeit nicht durchgeführt.

```
INPUT_SHAPE = (50,50,3)
BATCH = 32
EPOCHS = 40

layer_sizes = [8,16,32,64]
conv_layers = [1,2,3,4]

for layer_size in layer_sizes:
    for conv_layer in conv_layers:
        NAME = "{}-conv-{}-nodes-{}-dense-{}".format(conv_layer, layer_size, dense_layer, int(time.time()))

        tensorboard = TensorBoard(log_dir="logs2/{}".format(NAME))

        model = Sequential()
        model.add(Convolution2D(filters = layer_size, kernel_size = (3, 3),
                                padding = 'same',
                                kernel_regularizer=keras.regularizers.l2(0.001),
                                input_shape = INPUT_SHAPE))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size = (2, 2)))

        for l in range(conv_layer-1):
            model.add(Convolution2D(filters = layer_size, kernel_size = (3, 3),
                                    padding = 'same',
                                    kernel_regularizer=keras.regularizers.l2(0.001)))
            model.add(Activation('relu'))
            model.add(MaxPooling2D(pool_size = (2, 2)))

        model.add(Flatten())

        model.add(Dense(8))
        model.add(Activation('softmax'))

        model.compile(loss='binary_crossentropy',
                      optimizer='adam',
                      metrics=['accuracy']
                     )

        history = model.fit(train_img_norm, train_label_binaer,
                            batch_size = BATCH,
                            epochs = EPOCHS,
                            validation_split = 0.3,
                            verbose = 1,
                            callbacks = [tensorboard])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 50, 50, 32)        896
_________________________________________________________________
activation_1 (Activation)    (None, 50, 50, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 25, 25, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 25, 25, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 25, 25, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 36872
_________________________________________________________________
activation_3 (Activation)    (None, 8)                 0
=================================================================
Total params: 47,016
Trainable params: 47,016
Non-trainable params: 0
_________________________________________________________________
```

Model Zusammenfassung

Natural Images

Zürcher Hochschule
für Angewandte Wissenschaften

Vorhersage

Confusion Matrix

```
[[422   5   9   2   0   0   0   0]
 [  3 566   6   2   0   0   0   0]
 [  6  17 434  37   5   0   0   0]
 [ 18  15 155 241  12   0   1   2]
 [  2   8  15   9 478   0   7   0]
 [  1   0   0   2   0 618   0   0]
 [  0   0   1   1   0   0 460   0]
 [  0   0   3   4   0   0   0 573]]
```

Besonders bei der Kategorie Cat und Dog gibt es falsche Vorhersagen.

Einiger Testbilder aus der Vorhersage

Legende
0-airplane
1-car
2-cat
3-dog
4-flower
5-fruit
6-motorbike
7-person

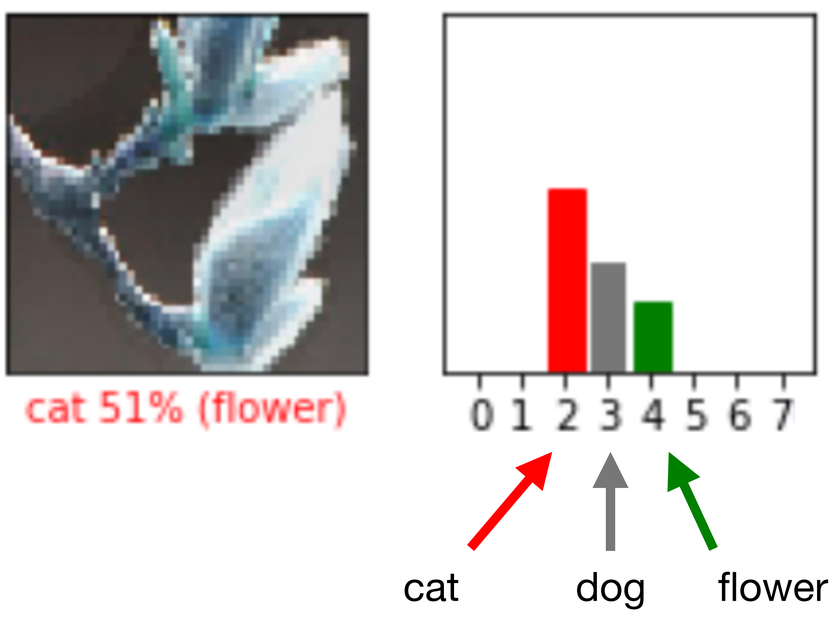
Natural Images

Zürcher Hochschule
für Angewandte Wissenschaften

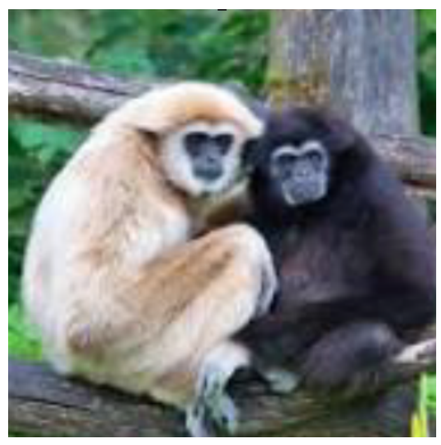
Ergebnisse und Erkenntnisse

In der Confusion Matrix ist zu erkennen, dass das Modell vor allem bei der Vorhersage von Katze und Hund oft Schwierigkeiten hat. Aber auch bei einigen Blumen kommt es zu Fehlinterpretationen. Als zusätzlichen Test wurde ein Blumenbild dem Modell VGG-16 übergeben. VGG-16 hat zu einer Wahrscheinlichkeit von 60% ein Gibbon vorhergesagt - die Vorhersage scheint also nicht ganz so einfach zu sein ...
Überraschend war für uns, dass es nicht immer ein kompliziertes und grosse Modell sein muss, mit zwei bis drei Layer lassen sich auch schon gute Ergebnisse von bis zu 98% erhalten.

Unser Modell



VGG-16



Bei dem Blumenbild (links) hat das Modell VGG-16 ein Gibbon vorhergesagt.

```
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
from keras.applications.vgg16 import VGG16

model = VGG16()
image = load_img(file, target_size=(224, 224))
image = img_to_array(image)
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
image = preprocess_input(image)
yhat = model.predict(image)
label = decode_predictions(yhat)
label = label[0][0]
print('%s (%.2f%%)' % (label[1], label[2]*100))

Ergebnis: gibbon (60.42%)
```

Eingereicht von André Wethmar, Roberto Ringenberg und Jan Schüpbach