ARTICIAL INTELLIGENCE: IMAGE RECOGNITION – CLASSIFICATION – REGRESSION – FEATURES EXTRACTION

1. Our Domain

Deep Learning, an effective part of Artificial Intelligence, not only makes it possible to draw serious conclusions from massive data, but also to make serious developments in science and technology. Among them, there are self-supervised technical systems, forecast of natural phenomena not allowing direct modeling and field experiments etc. This is can be done fundamental ability of deep learning to identify significant data properties ina hierarchical, “fractal” manner. Herewith, the depth of the hierarchy is limited by nothing (except for available computing resources), given an ML model of a relevant complexity.

Such models are neural networks, that are distributed computing systems analogous to the biological brain in two aspects:

- the ability to comprehend the information, to learn from data is achieved by both systems through training,
- the result of learning is fixed as strength of interconnections (synaptic weights) between neurons (functional elements).

At the same time, neural networks have a more distinct structuredness in comparison with the brain: the neurons are organized into layers so that fixed interrelations are formed between neurons of adjacent layers only.

Communications mutuality is ensured by two basic learning phases: direct (feed forward) and reverse (back propagation) [4]. The number of layers largely determines the “hierarchical depth” of the input data properties that the network is able to resolve

For neurons of adjacent layers, either all their possible pairs can be connected, as in classical (fully connected) layers, or fewer connections are established, as in convolutional layers. Such layers are characteristic of convolutional networks, one of which is the network used in the problem of image analysis described below.

The choice of the layer type is typically determined by the type of a task, as well as a compromise between the accuracy of the neural network and the amount of computations required.

2. Investigation of Children's Drawings

2.1. What is investigated

The set of children's pics collected as a result of psychological research and available at *http://pics.psych.stir.ac.uk*, is jpeg files, whose names contain a number of categorical and quantitative information about the pic and its author.
For example, the Fig. 1 left drawing is taken from file p3-67w-6.7f.jpeg, which means:

- age category - p3,
- age - 6 years, 7 months,
- author - girl,
- who is drawn - girl.

The right pic file is nu-21s-4.5m.jpeg, therefore:

- age category – nursery,
- age - 4 years, 5 months,
- author - boy (self picture),
- who is drawn - boy.

Figure 1. Two pieces of the children's drawings collection

2.2. What makes the problem special and challenging

2.2.1. Uniqueness of drawings

It is known that for a successful image recognition the image should be composed of more or less regular elements, or features. For example, human faces in all their diversity have easily recognizable details: eyes, noses, etc. Bodies are typically identified by a head, limbs, etc.

However, this is not the case of the children's pictures. Fig. 1 shows how few common, regular and recognizable features there are on a casually chosen pair of the collection. This is the situation with almost all the rest of the drawings. They are unique, reflect the individuality and personal imagination of the kids. To successfully solve the problems of a network training, it was thus necessary to carefully develop its architecture and training protocol trying a large number of parameters, options, and optimization possibilities. This is described in more detail in Section 3.

2.2.2. Multiplicity of factors

Section 2.1 revealed 4 types of categories, as well as one numeric parameter. The problem was solved in two independent versions:

- classification by categories,
- regression prediction of age as a quantitative parameter.

The problem of multifactor classification was solved using the Hot Encoding. Several groups of categories were aggregated to form a single group in such a way that each category of this group corresponded to a certain unique combination of the initial categories. For this, Keras, the Python deep learning framework was used.

It was made it possible to teach the network either on all four groups, or on any or some of them (1-3 groups). In the case of all four categorical groups, the general class consisted of 186 categories.

Note that all the options (choice of all or part of the categories) gave close accuracies of classification. This is an interesting point: basically, one could expect that a pic is most accurately identified by the complete set of categories. In fact, it turned out that each of four categories is self-sufficient and able to classify a drawing (see Table 1).

2.2.3. Small dataset

Only 338 drawings ... In practice, a successful learning requires at least several thousand elements.

One of the ways to solve such a problem is an augmentation of data by stochastic rotations, displacements, horizontal and vertical flapping of the initial pictures. This way, using the Keras ImageGenerator utility, the set of drawings was increased up to 17.500 pics. Similarly, we got a validation dataset of 3.500 drawings.

2.2. The Network Structure and Learning Protocol

2.2.1. The network architecture

The key to a successful solution of the problem was building a right network. The developed convolutional neural network was named DeeperNet (Fig. 2). It is deeper than the well-known LeNet network that has two pairs of the convolutional and pooling layers and did not give the solution to the problem involved.

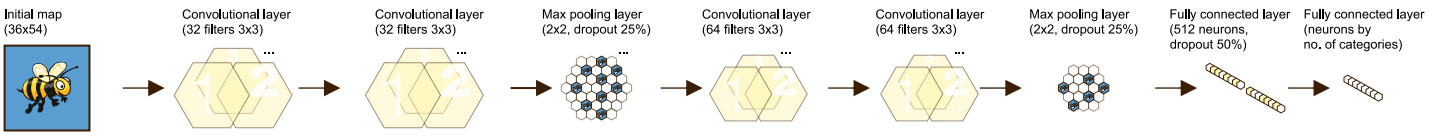

Figure 2. DeeperNet – a Convolutional Neural Network constructed for the solution of the problem

DeeperNet, instead, has two blocks, each consisting of a pair of convolutional layers and one max pool layer. These blocks are followed by two traditional (fully connected) layers. Another feature of DeeperNet is use of dropout in the max pool layers to fight overfitting.

The Python – Keras code constructing the network is shown in Appendix A.

Dropout is freezing, i.e., exclusion of a certain part of neurons randomly selected for each epoch (a cycle of work with a complete set of learning dataset) from the training. If we use an analogy with the biological brain, each new learning cycle is made with the "working" part of the brain, somewhat different from the previous portion. This makes the brain have slightly different experience. Thus, an ensemble of learners (within the framework of one model) arises — just as in well-known ensembleing methods (bagging, boosting etc.). The solution of our problem confirmed that dropout really effectively compensates the tendency of convolutional neural networks to get overfit if their topology is rather complicated.

Many researchers note that super-results can be achieved with a reasonable combination of dropout and dropconnect (when not a part of neurons but a certain part of connections between neurons is isolated). This combination is used in the tasks we are solving currently.

3.2. Organization of the learning process

3.2.1. 10-fold cross validation

A traditionally used procedure in which the entire training data set is split in 10:1 ratio so that 9/10 of the data set is used for training, and the rest for validation. As a result, 9 folds are created so that each component is used for validation once and only once. The splitting is repeated many times during the learning process.

There was used the option of weighted cross-validation (stratified cross validation). For that, the Python sklearn package was used (see Appendix B).

3.2.2. Early stopping

Usually, a machine learning task is something more complicated than bare achieving maximum learning accuracy. As is well known, starting from a certain level of the learning accuracy reached, the generalization, i.e., the accuracy being obtained on data that does not belong to the training set, begins to worsen. This is illustrated in Figure 4. Highlighted is the point where the validation error no more diminishes. So it is necessary to interrupt the training.

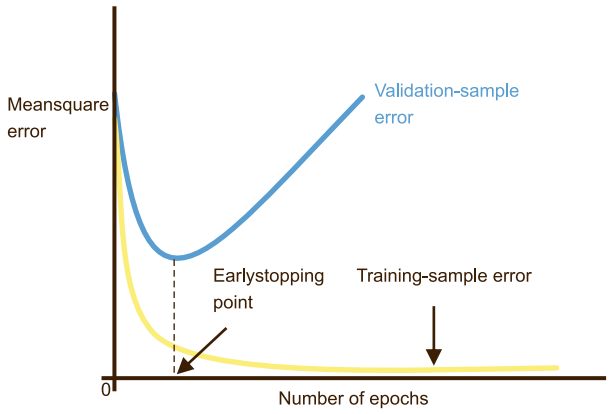

Figure 3. In the process of learning, generalization gets worse.
Early stopping is thus needed.

In practice stopping occurs after detection of a sequence of epochs with a monotonous raise of the error. The length of this sequence is controlled by a parameter called tolerance. In our case, it was set equal to 7 (see Appendix A). The learning process with early stopping is illustrated in Fig. 4.

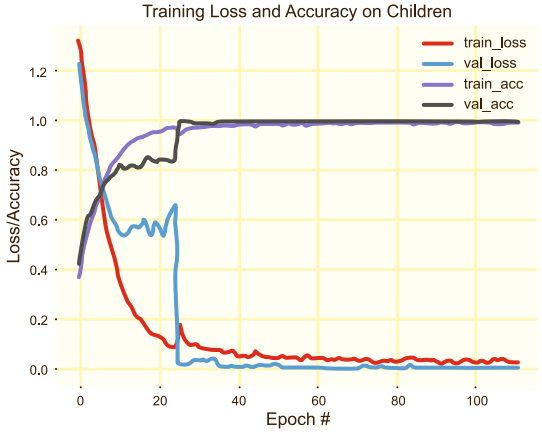

Figure 4. Training and validation:
change of errors and accuracy (portions of correctly guessed categories values).

3.2. Results

So, on the training set of 17.500 and the test set of 3.500 drawings, two tasks were solved:

Regression. The prediction of age as a quantitative parameter gave accuracy of 87.05 %.

Classification. As pointed above, making use of one, two, three, or all four factors to classify gave comparable accuracy (percentage of properly guessed categories), as illustrated in Table 1.

Note that the reached level of accuracy was kept the same with decrease of the training set (by random sampling) up to 40% of the original its size.

Table 1. Classification accuracy with various sets of categories.

Categories included	Accuracy
'age', 'who drew', 'who is drawn', 'age category'	85.58%
'who drew', 'who is drawn', 'age category'	77.80%
'who is drawn', 'age category'	83.72%
'age category'	88.54%

4. Conclusion. About new tasks

Finally, describe the extension of this task on new presently solved problems.

In the described task, involving a small dataset, each drawing is "labeled", i.e., initially classified.

In the case of data sets with tens of thousands and much more elements, it is obviously not possible to "tag" each element. A regular machine learning approach in such cases is unsupervised learning, i.e., self-categorization of the data.

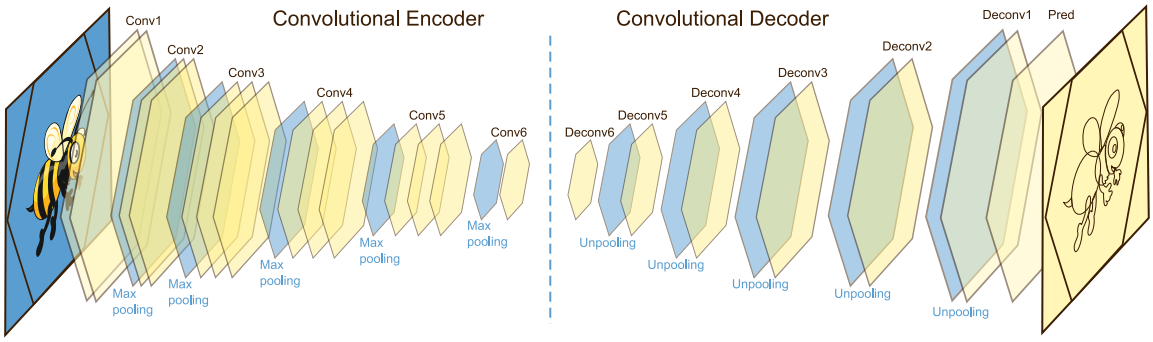

Figure 5. Convolutional encoder-decoder network.

With regard to the image analysis and convolutional networks, this is implemented, for example, in the form of encoding-decoding of each individual image using an hourglass-shaped network - like the one shown in Fig. 5. The layer in the narrowest middle part of the network contains a small amount of abstract information about the image, its most essential components. They can be interpreted as the image categories.

Presently we test a similar encoding - decoding network on a large database of images. In addition to other projects, it will be very interesting to apply this network to the presented children's drawings disregarding their original categories - and to compare the new solution with that described here.

Appendices

A.Constructing the DeeperNet

```
"""
This constructs a fairly deep
convolutional network
"""
from keras.models import Sequential
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation, Dropout
from keras.layers.core import Flatten
from keras.layers.core import Dense
from keras import backend as K

class Deeper:
   @staticmethod
   def build(width, height, depth, classes_no):
      model = Sequential ()

      inputShape = (height, width, depth)
      # if we are using "channels first", update the input shape
      if K.image_data_format () == "channels_first":
         inputShape = (depth, height, width)

      model.add (Conv2D (32, (3, 3), padding='same',
                  input_shape=inputShape))
      model.add (Activation ('relu'))
      model.add (Conv2D (32, (3, 3), padding='same'))
      model.add (Activation ('relu'))
      model.add (MaxPooling2D (pool_size=(2, 2)))
      model.add (Dropout (0.25))
      model.add (Conv2D (64, (3, 3), padding='same'))
      model.add (Activation ('relu'))
      model.add (Conv2D (64, 3, 3))
      model.add (Activation ('relu'))
      model.add (MaxPooling2D (pool_size=(2, 2)))
      model.add (Dropout (0.25))
      model.add (Flatten ())
      model.add (Dense (512))
      model.add (Activation ('relu'))
      model.add (Dropout (0.5))
      model.add (Dense (classes_no))
      model.add (Activation ('softmax'))
      return model
```

B.Configuring the Network Learning Process

```
from keras import optimizers
from keras.callbacks import EarlyStopping
import extractTargets
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from sklearn.model_selection import StratifiedKFold
from keras.preprocessing.image import img_to_array
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from CNN.Custom import Custom
from CNN.Deeper import Deeper
from imutils import paths
import numpy as np
import random
import cv2
import os
import time

start = time.time()
matplotlib.use("Agg")

# initialize number of epochs to train for,
# initial learning rate,
# and batch size
EPOCHS = 25
INIT_LR = 1e-3
BS = 128
SAMPLE_PORTION = .05
MAX_PIXEL_INTENSITY = 255.0
WIDTH = 36
HEIGHT = int(WIDTH*8/5)
```

```
# initialize data and targets
print("[INFO] loading images...")
data = []
targets = []
# cats = ["age", "who drew", "who is drawn", "age category"]
cats = ["who drew", "who is drawn", "age category"]

n_splits=10

# grab the image paths and randomly shuffle them
imagePaths = sorted(list(paths.list_images("images")))
random.seed(42)
random.shuffle(imagePaths)

# grab the image paths and randomly pick a sample of them
imagePaths = sorted(list(paths.list_images("images")))
imagePaths = np.random.choice(imagePaths, int(len(imagePaths)
                * SAMPLE_PORTION), replace=False)

for imagePath in imagePaths:
    # load the image, pre-process it,
    # and store it in the data list
    image = cv2.imread(imagePath, cv2.IMREAD_GRAYSCALE)
    image = cv2.resize (image, (WIDTH, HEIGHT))
    image = img_to_array(image)
    data.append(image)
    # extract the class target from the image path
    target = extractTargets.extractTargets(
        os.path.splitext(os.path.basename(imagePath))[0], cats)
    targets.append(target)

# scale the raw pixel intensities to the range [0, 1]
data = np.array(data, dtype="float") / MAX_PIXEL_INTENSITY

# preparing ourselves for 10-fold stratified cross validation
skf = StratifiedKFold(n_splits)
split = skf.split(data, targets)

# encode class values as integers
encoder = LabelEncoder()
encoder.fit(targets)
targets = encoder.transform(targets)
initial_targets = targets
targets = to_categorical(targets)

print("[INFO] compiling model...")
model = Deeper.build(width=WIDTH, height=HEIGHT,
                depth=1, classes_no=targets.shape[1])
sgd = optimizers.Adam (lr=0.001, beta_1=0.9,
                beta_2=0.999, decay=0.0)

model.compile(loss="categorical_crossentropy", optimizer=sgd,
            metrics=["accuracy"])

fold = 1
history_loss = []
history_val_loss = []
history_acc = []
history_val_acc  = []

for train_index, test_index in split:
    print("Fold no. " + str(fold))
    fold = fold + 1
    X_train, X_test = data[train_index], data[test_index]
    y_train = targets[train_index]
    y_test = targets[test_index]
    initial_y_test = initial_targets[test_index]

    # construct the image generator for data augmentation
    aug = ImageDataGenerator()
    H = model.fit_generator(aug.flow(X_train,
            y_train, batch_size=BS),
            validation_data=(X_test, y_test),
            steps_per_epoch=len(X_train) // BS,
            epochs=EPOCHS, verbose=1,
            callbacks=[EarlyStopping(monitor="val_loss",
            patience=7, mode="auto")])
```

```
    history_loss = history_loss + H.history["loss"]
    history_val_loss = history_val_loss + H.history["val_loss"]
    history_acc = history_acc + H.history["acc"]
    history_val_acc = history_val_acc + H.history["val_acc"]


# serialize model to JSON
model_json = model.to_json()
with open("models/children.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("models/children.h5")
print("Saved model to disk")
```