Using CNNs to classify people in Famous48 dataset - Subject 12c

Jakub Czermański
193105

Paweł Blicharz
193193

02.05.2024

Contents

Contents

1 Introduction

Lorem ipsum

Data description

famous48 is a set of example images contained faces of 48 famous persons like sportsmens, politicians, actors or television stars. It was divided into 3 files: x24x24.txt, y24x24.txt, z24x24.txt, each containing 16 personal classes.

Attributes description:

- a1 - face containing flag: (1-with face, 0-without face)
- a2 - image number in current class (person) beginning from 0
- a3 - class (person) number beginning from 0
- a4 - sex (0 - woman, 1 - man)
- a5 - race (0- white, 1 - negro, 2 - indian, ...)
- a6 - age (0 - baby, 1 - young, 2 - middle-age, 3 - old)
- a7 - binokulars (0 - without, 1 - transparent, 2 - dark)
- a8 - emotional expression (not state!) (0 - sad, 1 - neutral, 2 - happy)

Note: Full code can be found in *notebook_keras.ipynb*. Due to limited space, we provide here only the most important code.

2 Libraries used

Firstly, we import all our libraries:

```
import numpy as np
import tensorflow as tf
from tensorflow import keras

from keras import layers
from keras import regularizers
from keras import backend as K
from keras import Sequential, Input
from keras.optimizers import SGD, Adam
from keras.losses import categorical_crossentropy
from keras.callbacks import LearningRateScheduler

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

import math
```

```python
from functools import partial

from utils.constants import CLASSES, IMAGE_SIZE
from utils.load_dataset import load_dataset
```

3 File loading

This is our code for loading files and we will not change it throughout our journey:

```python
CLASSES = 48
IMAGE_SIZE = 24

def read_file(filename):
    with open(filename, 'r') as file:
        file.readline() # we skip the first line as it is not needed
        number_of_pixels = int(file.readline())

        features = []
        labels = []

        for line in file.readlines():
            elements = line.split()

            # add features
            pixels = np.array(elements[:number_of_pixels], dtype=float)
            pixels = np.reshape(pixels, [IMAGE_SIZE, IMAGE_SIZE])
            features.append(pixels)

            # add labels
            labels.append(elements[number_of_pixels+2])

        features = np.array(features)
        labels = np.array(labels, dtype=int)

    return features, labels


def load_dataset(directory_path):
    X_0, y_0 = read_file(f'{directory_path}/x24x24.txt')
    X_1, y_1 = read_file(f'{directory_path}/y24x24.txt')
    X_2, y_2 = read_file(f'{directory_path}/z24x24.txt')

    X = np.concatenate((X_0, X_1, X_2))
    y = np.concatenate((y_0, y_1, y_2))

    N_TRAIN_EXAMPLES=int(len(X) * 0.8)
```

```
    N_TEST_EXAMPLES=len(X) - N_TRAIN_EXAMPLES

    X_train, X_test, y_train_raw, y_test_raw = train_test_split(X, y,
                                                                train_size=N_TRAIN_EXAMPLES,
                                                                test_size=N_TEST_EXAMPLES,
                                                                random_state=42)

    y_train = np.zeros((y_train_raw.shape[0], CLASSES))
    y_test = np.zeros((y_test_raw.shape[0], CLASSES))

    for i, value in enumerate(y_train_raw):
        y_train[i][value] = 1

    for i, value in enumerate(y_test_raw):
        y_test[i][value] = 1

    return X_train, X_test, y_train, y_test


DIRPATH = './data'
X_train, X_test, y_train, y_test = load_dataset(DIRPATH)
```

4 Testing

Here are our all attempts at finding the best model. We tried out different architectures and hyperparameters to achieve that goal.

4.1 AlexNet

Firstly, we implemented AlexNet model (`https://papers.nips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html`) which won the 2012 ILFRC challenge. Here is our model in python code, after downscaling it and changing some parameters:

```
conv_regularizer = regularizers.l2(0.0006)
dense_regularizer = regularizers.l2(0.01)

DefaultConv2D = partial(tf.keras.layers.Conv2D, kernel_size=3, padding="same",
                        activation="relu", kernel_regularizer=conv_regularizer)

model = keras.Sequential([
    Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 1)),
    DefaultConv2D(96),
    layers.MaxPooling2D(pool_size=3, strides=2),

    tf.kears.layers.Dropout(0.3),
```

```
    DefaultConv2D(256, kernel_size=5),
    tf.keras.layers.MaxPooling2D(pool_size=3, strides=2),

    tf.keras.layers.Dropout(0.4),
    DefaultConv2D(384),
    tf.keras.layers.Dropout(0.5),
    DefaultConv2D(384),
    tf.keras.layers.MaxPooling2D(pool_size=3, strides=2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.6),
    tf.keras.layers.Dense(384, activation='relu',
                          kernel_regularizer=dense_regularizer),
    tf.keras.layers.Dense(CLASSES, activation='softmax')
])
```

Hyperparameters & methods used:

- *optimizer*: Adam, with *learning rate* = 0.001
- *epochs* = 250
- *batch size* = 200
- *validation set size* = 20% of the training set
- *shuffle* the training data before each epoch
- L2 regularization for convolutional layers: $l = 0.0006$
- L2 regularization for the dense layer: $l = 0.01$

Results: (rounded to 3 decimal places)

- train set accuracy: 0.896
- train loss: 0.775
- validation set accuracy: 0.81
- validation set loss: 1.115
- test set accuracy: **0.8**
- test set loss: 1.137

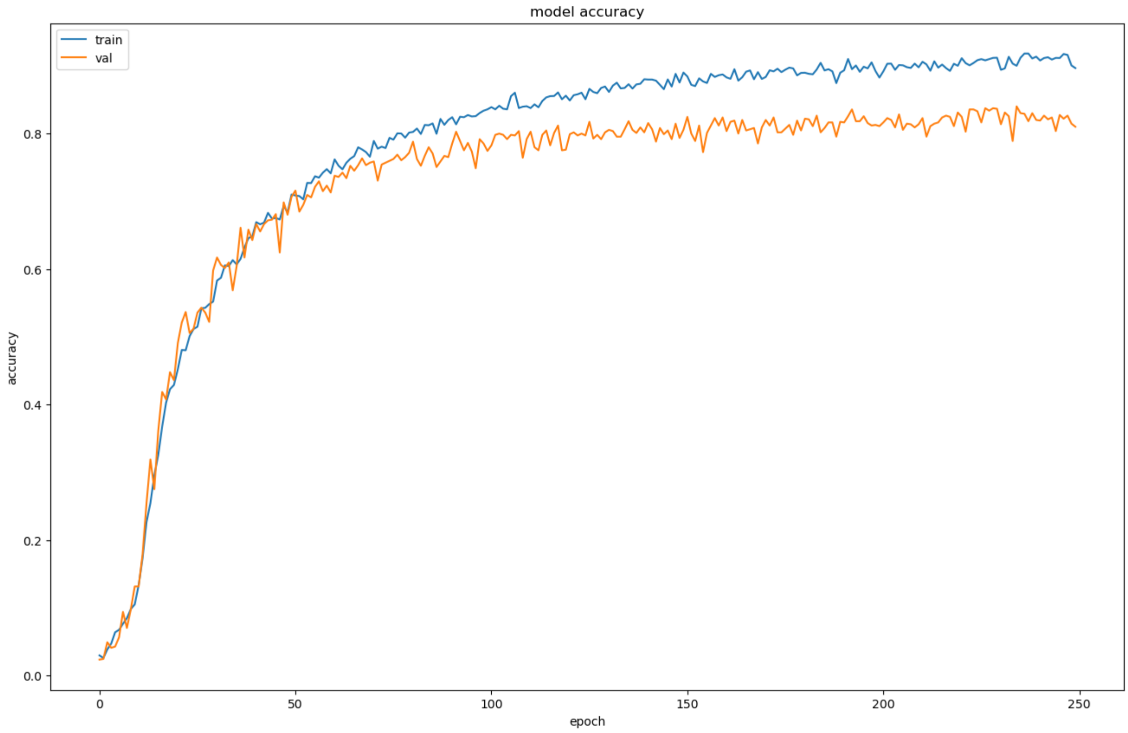

Figure 1: Model accuracy for AlexNet architecture - training & validation sets

Conclusions:

We decided that 80% is not enough so we did not tune hyperparameters.

4.2 LeNet-5

Next, we decided to used LeNet-5 architecture. a

4.2.1 Base model

```
conv_regularizer = regularizers.l2(0.0006)
dense_regularizer = regularizers.l2(0.01)

DefaultConv2D = partial(tf.keras.layers.Conv2D, kernel_size=5, padding="same",
                        activation="tanh", kernel_regularizer=conv_regularizer)

model = Sequential(
  [
    Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 1)),
    DefaultConv2D(6),
    layers.MaxPooling2D(pool_size=2, strides=2),

    layers.Dropout(0.45),
```

```
    DefaultConv2D(16),
    layers.MaxPooling2D(pool_size=2, strides=2),

    layers.Dropout(0.3),
    DefaultConv2D(120),

    layers.Flatten(),
    layers.Dropout(0.15),
    layers.Dense(84, activation=activation_def,
                 kernel_regularizer=dense_regularizer),
    layers.Dense(CLASSES, activation='softmax'),
  ]
)
```

4.2.2 Testing hyperparameters

Attempt 1
We select all hyperparameters by hand

Hyperparameters & methods used:

- *optimizer*: Adam, with *learning rate* = 0.001
- *epochs* = 100
- *batch size* = 200
- *validation set size* = 20% of the training set
- *shuffle* the training data before each epoch
- L2 regularization for convolutional layers: $l = 0.0006$
- L2 regularization for the dense layer: $l = 0.01$

Results:

- train set accuracy: 0.805
- train loss: 1.07
- validation set accuracy: 0.81
- validation set loss: 1.08
- test set accuracy: **0.81**
- test set loss: 1.08

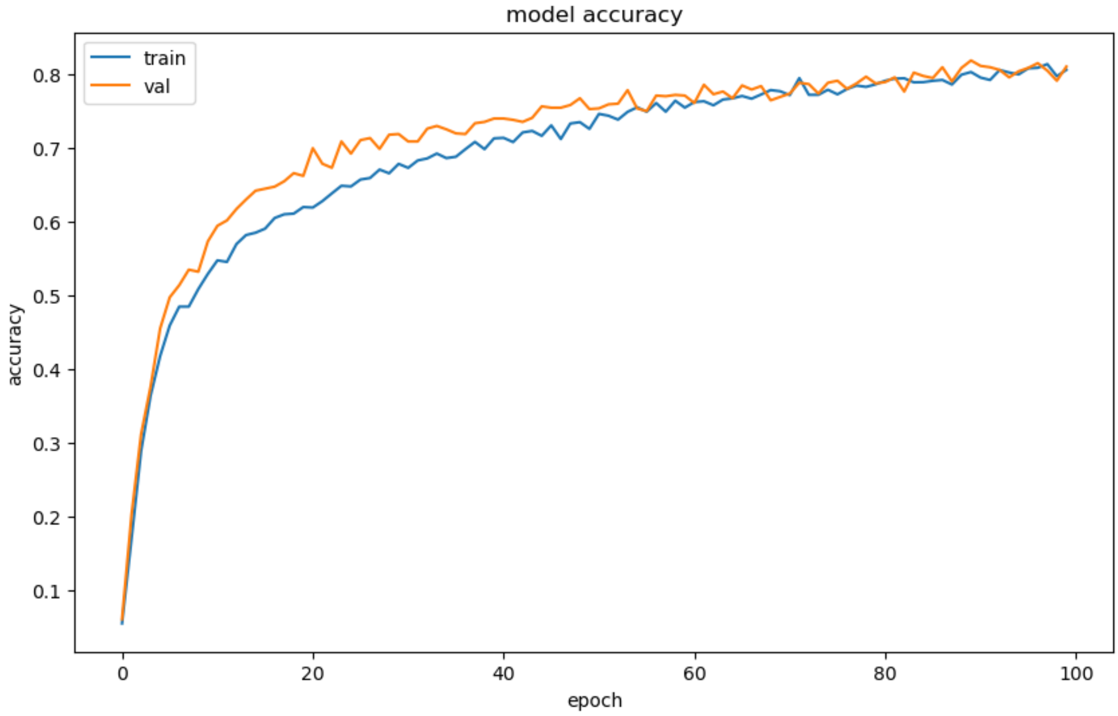

Figure 2: Model accuracy for LeNet-5 architecture - training & validation sets

Attempt 2

We used optuna to find hyperparameters for regularization, introduced decaying learning rate and increased number of epochs to 1000.

Hyperparameters & methods used:

- *optimizer*: Adam, with starting *learning rate* = 0.001, and decreasing by 50% each 150 epochs
- *epochs* = 1000
- *batch size* = 200
- *validation set size* = 20% of the training set
- *shuffle* the training data before each epoch
- L2 regularization for convolutional layers: $l \approx 0.00061$
- L2 regularization for the dense layer: $l \approx 0.0301$

Results:

- train set accuracy: 0.981
- train loss: 0.337
- validation set accuracy: 0.878
- validation set loss: 0.651
- test set accuracy: **0.892**
- test set loss: 0.629

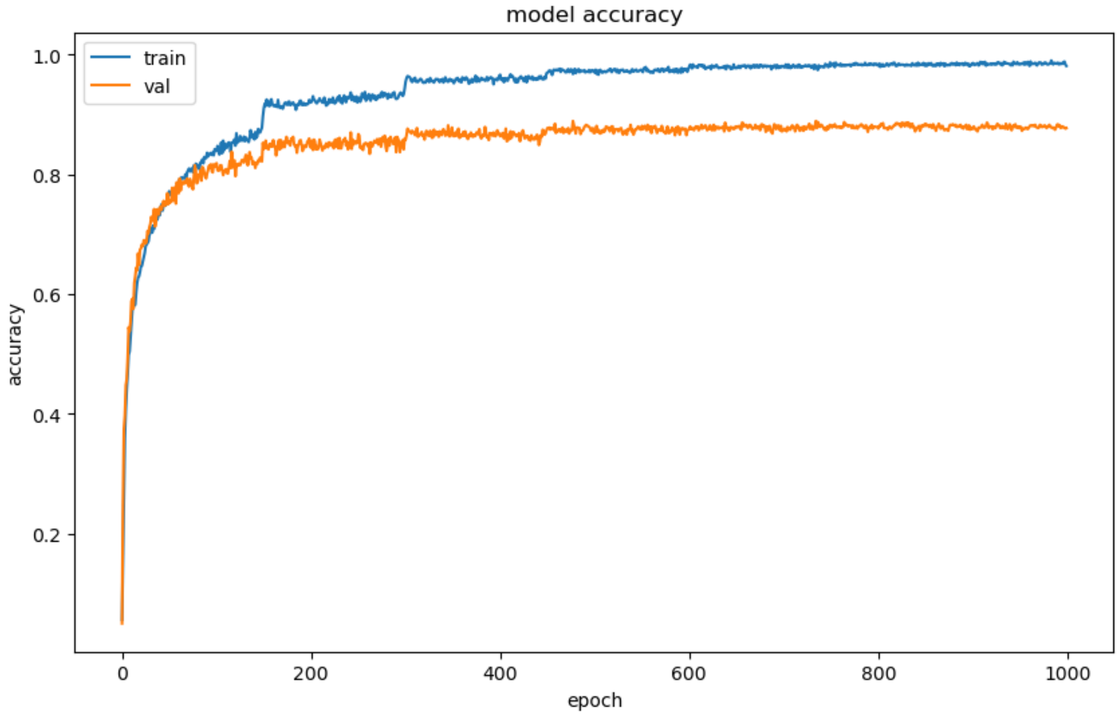

Figure 3: Model accuracy for LeNet-5 with optimized hyperparameters

Conclusions:

The model after 20 epochs starts overfitting and validation set's accuracy improves very slowly.
(to-do)

5 Final model

As our final model we decided to have Lenet-5 because ...

[1]