

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

Jakub Czernański  
193105

Paweł Blicharz  
193193

02.05.2024

# Contents

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Libraries used</b>	<b>1</b>
<b>3</b>	<b>File loading</b>	<b>2</b>
<b>4</b>	<b>Testing</b>	<b>3</b>
4.1	Architectures . . . . .	3
4.1.1	AlexNet . . . . .	3
4.1.2	LeNet-5 . . . . .	5
<b>5</b>	<b>Final model</b>	<b>6</b>

# 1 Introduction

Lorem ipsum

## Data description

famous48 is a set of example images contained faces of 48 famous persons like sportsmen, 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:

```
1 import numpy as np
2 import tensorflow as tf
3 from tensorflow import keras
4
5 from keras import layers
6 from keras import regularizers
7 from keras import backend as K
8 from keras import Sequential, Input
9 from keras.optimizers import SGD, Adam
10 from keras.losses import categorical_crossentropy
11 from keras.callbacks import LearningRateScheduler
12
13 from sklearn.model_selection import train_test_split
14
15 import matplotlib.pyplot as plt
16
17 import math
18 from functools import partial
```

### 3 File loading

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

```
1 CLASSES = 48
2 IMAGE_SIZE = 24
3
4 def read_file(filename):
5
6     with open(filename, 'r') as file:
7         file.readline() # we skip the first line as it is not needed
8         number_of_pixels = int(file.readline())
9
10        features = []
11        labels = []
12
13        for line in file.readlines():
14            elements = line.split()
15
16            # add features
17            pixels = np.array(elements[:number_of_pixels], dtype=float)
18            pixels = np.reshape(pixels, [IMAGE_SIZE, IMAGE_SIZE])
19            features.append(pixels)
20
21            # add labels
22            labels.append(elements[number_of_pixels+2])
23
24        features = np.array(features)
25        labels = np.array(labels, dtype=int)
26
27        return features, labels
28
29
30 X_0, y_0 = read_file('./data/x24x24.txt')
31 X_1, y_1 = read_file('./data/y24x24.txt')
32 X_2, y_2 = read_file('./data/z24x24.txt')
33
34 X = np.concatenate((X_0, X_1, X_2))
35 y = np.concatenate((y_0, y_1, y_2))
36
37 X_train, X_test, y_train_raw, y_test_raw = train_test_split(X, y,
38                                                                train_size=N_TRAIN_EXAMPLES,
39                                                                test_size=N_TEST_EXAMPLES,
40                                                                random_state=42)
41
42 y_train = np.zeros((y_train_raw.shape[0], CLASSES))
43 y_test = np.zeros((y_test_raw.shape[0], CLASSES))
```

```

44
45 for i, value in enumerate(y_train_raw):
46     y_train[i][value] = 1
47
48 for i, value in enumerate(y_test_raw):
49     y_test[i][value] = 1

```

## 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 Architectures

#### 4.1.1 AlexNet

Firstly, we implemented AlexNet model ([https://papers.nips.cc/paper\\_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html](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:

```

1 conv_regularizer = regularizers.l2(0.0006)
2 dense_regularizer = regularizers.l2(0.01)
3
4 DefaultConv2D = partial(tf.keras.layers.Conv2D, kernel_size=3, padding="same",
5                         activation="relu", kernel_regularizer=conv_regularizer)
6
7 model = keras.Sequential([
8     Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 1)),
9     DefaultConv2D(96),
10    layers.MaxPooling2D(pool_size=3, strides=2),
11
12    tf.keras.layers.Dropout(0.3),
13    DefaultConv2D(256, kernel_size=5),
14    tf.keras.layers.MaxPooling2D(pool_size=3, strides=2),
15
16    tf.keras.layers.Dropout(0.4),
17    DefaultConv2D(384),
18    tf.keras.layers.Dropout(0.5),
19    DefaultConv2D(384),
20    tf.keras.layers.MaxPooling2D(pool_size=3, strides=2),
21
22    tf.keras.layers.Flatten(),
23    tf.keras.layers.Dropout(0.6),
24    tf.keras.layers.Dense(384, activation='relu',
25                          kernel_regularizer=dense_regularizer),

```

```

26     tf.keras.layers.Dense(CLASSES, activation='softmax')
27 ])

```

### 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

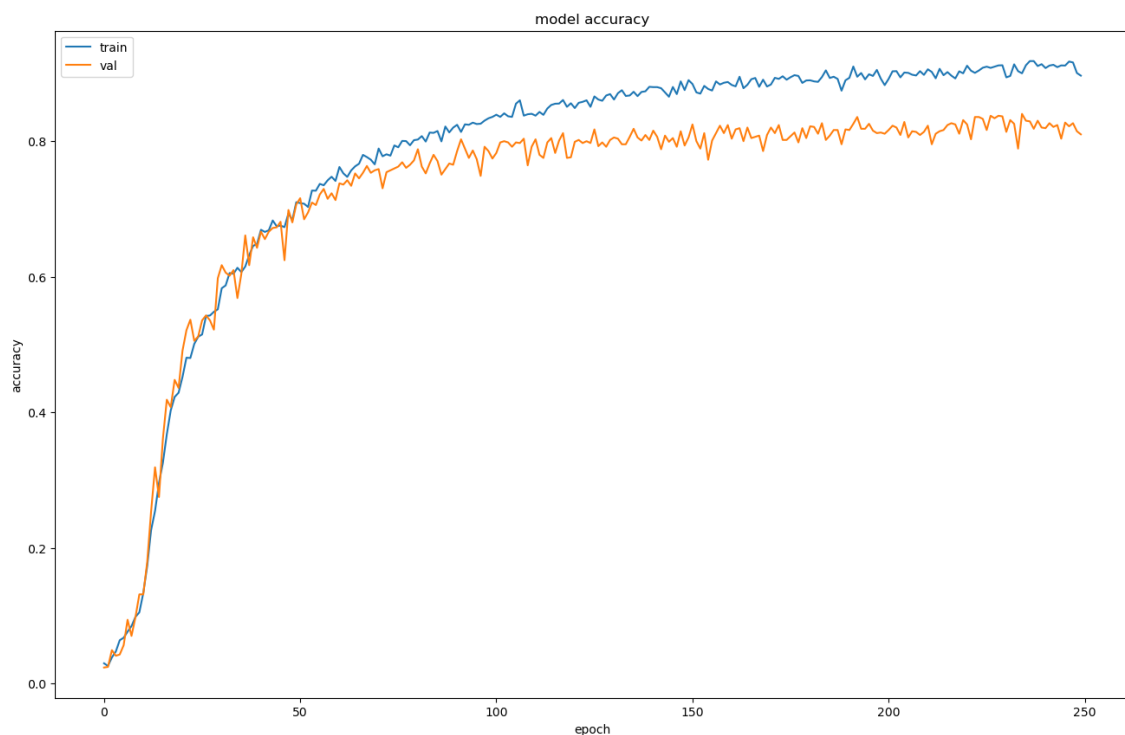


Figure 1: Model accuracy for AlexNet architecture - training & validation sets

## Conclusions:

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

### 4.1.2 LeNet-5

Next, we decided to use LeNet-5 architecture, presented below:

```
1 conv_regularizer = regularizers.l2(0.0006)
2 dense_regularizer = regularizers.l2(0.01)
3
4 DefaultConv2D = partial(tf.keras.layers.Conv2D, kernel_size=5, padding="same", acti
5                        kernel_regularizer=conv_regularizer)
6
7 model = Sequential(
8     [
9         Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 1)),
10        DefaultConv2D(6),
11        layers.MaxPooling2D(pool_size=2, strides=2),
12
13        layers.Dropout(0.45),
14        DefaultConv2D(16),
15        layers.MaxPooling2D(pool_size=2, strides=2),
16
17        layers.Dropout(0.3),
18        DefaultConv2D(120),
19
20        layers.Flatten(),
21        layers.Dropout(0.15),
22        layers.Dense(84, activation=activation_def, kernel_regularizer=dense_regularizer),
23        layers.Dense(CLASSES, activation='softmax'),
24    ]
25 )
```

## 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: (rounded to 3 decimal places)

- 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

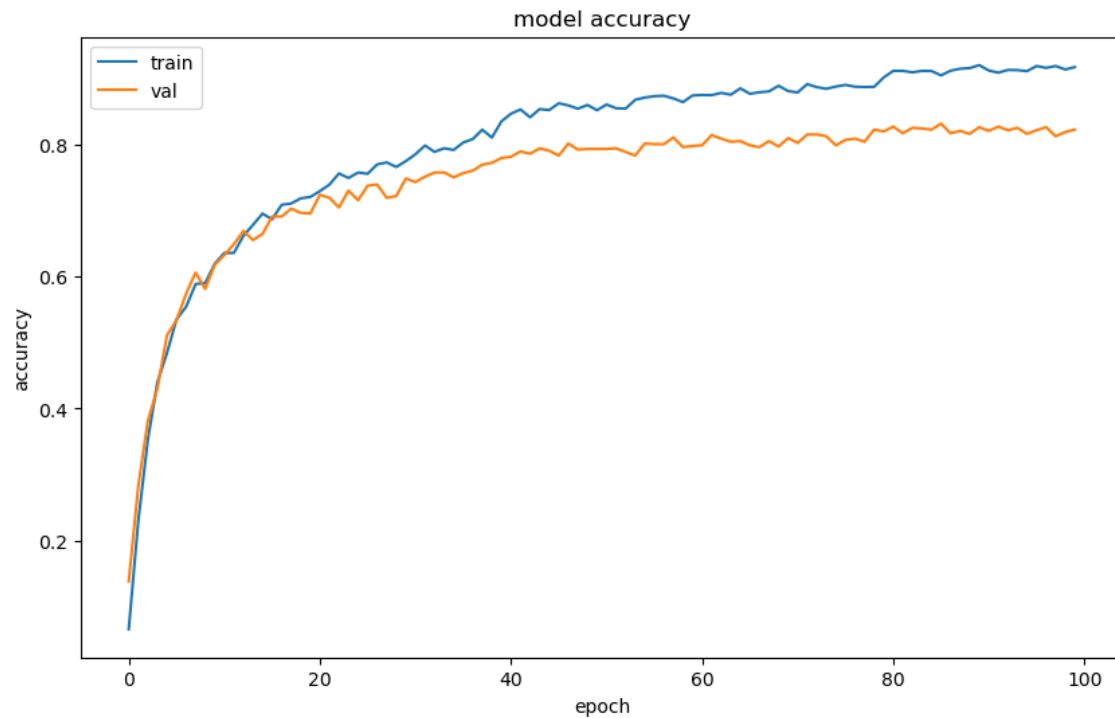


Figure 2: Model accuracy for LeNet-5 architecture - training & validation sets

## Conclusions:

The model after 20 epochs starts overfitting and validation set's accuracy improves very slowly.

## 5 Final model

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