

Facial detection and recognition in animated series

In this paper, I will describe an approach to detect and recognize faces extracted from “Life with Louie” animated series photos.

Task 1: Face detection

If we want to recognize faces from images, we first need to detect them and, later on, recognize the character that we have in that image. In this task we will focus on image detection, while in task2 we will talk more about the recognition part.

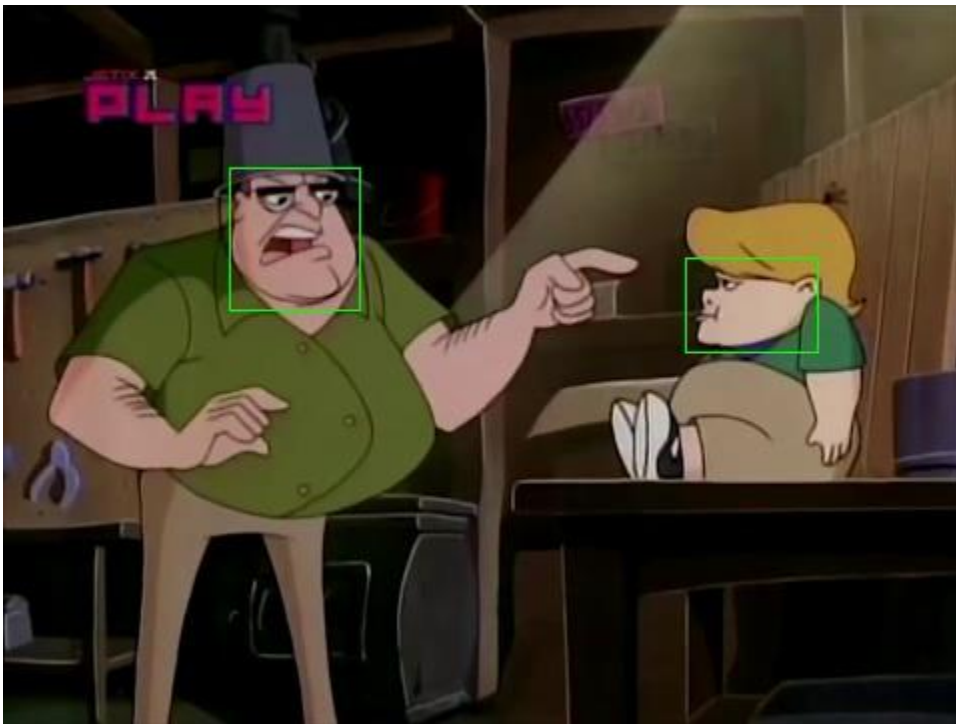
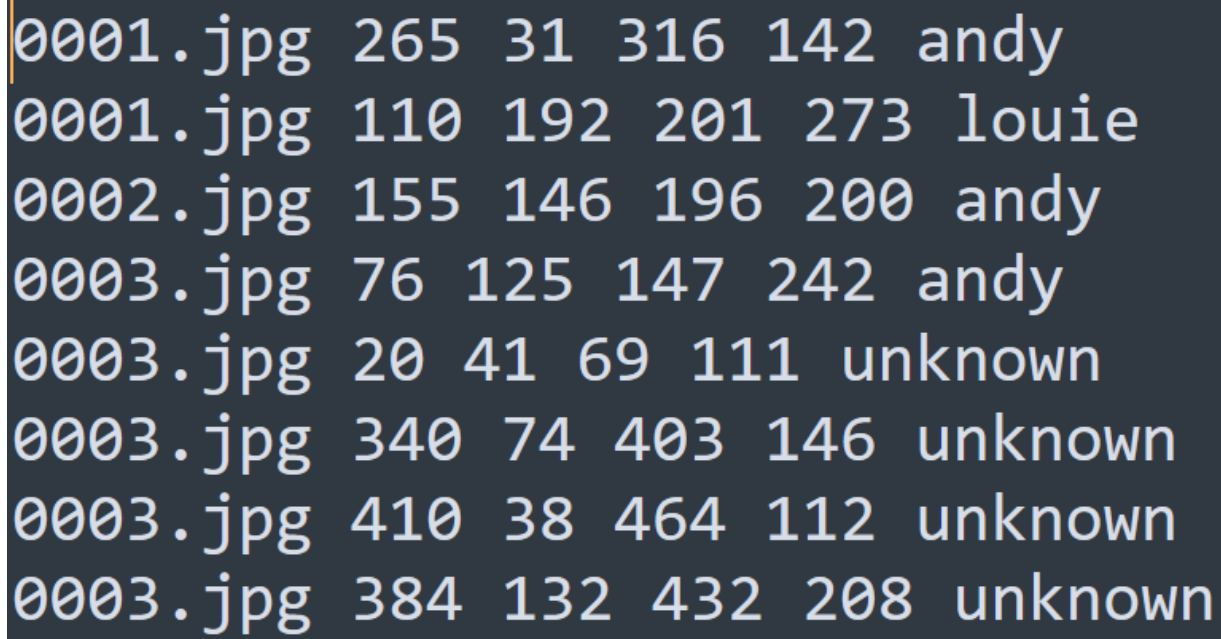


Fig 1. An image from the dataset and the faces desired to identify (marked by the green boxes)

Step 1: Generating the dataset

The algorithm receives .txt files that contain on each line 6 space separated words and numbers: a file name, four 4 numbers describing the top left and top right corner of a box inside the image who's name we received and the name of the character that is located there.



```
0001.jpg 265 31 316 142 andy
0001.jpg 110 192 201 273 louie
0002.jpg 155 146 196 200 andy
0003.jpg 76 125 147 242 andy
0003.jpg 20 41 69 111 unknown
0003.jpg 340 74 403 146 unknown
0003.jpg 410 38 464 112 unknown
0003.jpg 384 132 432 208 unknown
```

Fig 2. Visualization of the described input format

We will create the faces dataset (valid images dataset) by opening the photo identified by the name received and crop the box described by the two corners, and add it with the label “1” in the dataset. We found 14472 positive faces around the 4000 photos received in the dataset.

For the negative dataset we do not receive any information so we have to extract it ourselves. We will loop through the images and, for each photo, extract random generated patches of desired size. We will further check for each extracted patch (before adding it to the dataset) to not intersect any face from that photo to prevent adding a valid face and classifying it as invalid, in which case we will generate another patch random which we will also check. If in a certain photo we can't find any valid spot in the first 100 random generations, we will consider the photo does not contain any free spot, and search for one in another photo. After we generated the negative examples, we will add them to the big dataset with the label "0", marking them as not faces.

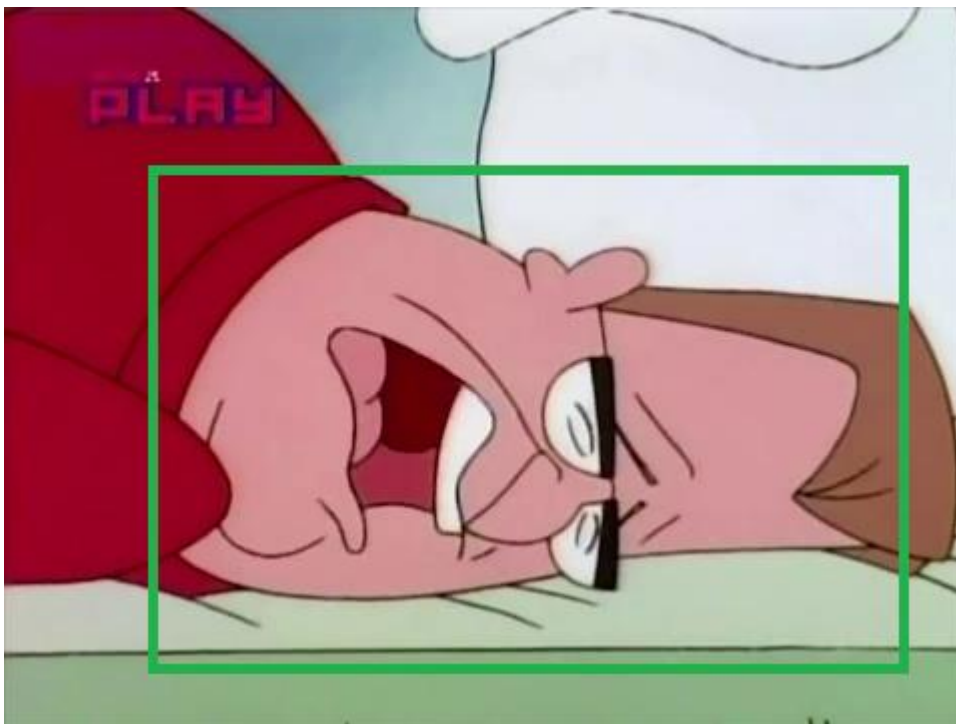


Fig 3. A photo from the dataset from which the algorithm didn't find any valid 96x96 square that does not intersect any face

Step 2: Training the model

Now that we have the training dataset, we must now train a model that will manage to predict if a photo contains or not a face. We will use a convolutional neural network (CNN) and train it on the dataset.

The best results were obtained on the structure described in Figure 4, having the learning rate 0.0001, with the Adam optimizer and the loss function Sparse Categorical Cross entropy after 10 epochs.

```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters = 32, kernel_size = (2, 2), activation = 'relu', input_shape = (box_size, box_size, 3)),
    keras.layers.MaxPooling2D(),

    keras.layers.Conv2D(filters = 64, kernel_size = (2, 2), activation = 'relu'),
    keras.layers.MaxPooling2D(),

    keras.layers.Conv2D(filters = 128, kernel_size = (2, 2), activation = 'relu'),
    keras.layers.MaxPooling2D(),

    keras.layers.Flatten(),

    keras.layers.Dense(256, activation = 'relu'),
    keras.layers.Dropout(0.25),

    keras.layers.Dense(64, activation = 'relu'),
    keras.layers.Dropout(0.25),

    keras.layers.Dense(32, activation = 'relu'),
    keras.layers.Dropout(0.25),

    keras.layers.Dense(16, activation = 'relu'),

    keras.layers.Dense(2, activation = 'softmax')
])

opt = keras.optimizers.Adam(learning_rate = 1e-4)
model.compile(optimizer = opt, loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
model.fit(training_examples, training_labels, epochs = 10, shuffle='True')
```

Fig 4. The structure of the neural network

Step 3: Generating predictions

Now that we have our model trained, all we have to do is to generate our predictions on the test dataset. For each image in the test dataset, we will extract patches of diverse sizes and will ask the model to predict scores for them. The patches that have a score greater than 0.8 (out of 1) will be considered as faces. After that, we will suppress the intersecting images of whom score is not a local maximum (by CNN's score) so we do have only one prediction per face.

```
#We get diverse size patches, resize them to box_size x box_size and adding them to the predict array
for saiz in range(132, 24,-9):
    for i in range(0, img.shape[0] - saiz, 4):
        for j in range(0, img.shape[1] - saiz, 4):

            patch = img[i : i + saiz, j : j + saiz, :]
            patch = cv.resize(patch,(box_size, box_size), interpolation = cv.INTER_AREA)

            patches.append(patch)
            squares.append([j, i, j + saiz, i + saiz])

patches = np.array(patches)
print(test_files[ii][-8:])

model_predictions = model.predict(patches)

predictions = []
scores = []

#If the score between 0 and 1 received from the photo is greater than 0.8, we consider it good
threshold = 0.8
for i in range(len(patches)):
    if model_predictions[i][1] > threshold:
        predictions.append(squares[i])
        scores.append(model_predictions[i][1])

nr_preds = len(predictions)
if nr_preds == 0:
    continue

predictions, scores = non_maximal_suppression(np.array(predictions), np.array(scores), img.shape)
```

Fig 5. The described algorithm for each photo from the test dataset

Task 2: Face recognition

Now that we know (at least theoretically) where the faces are, we can try identifying who are the persons with that faces. We will classify the faces of Andy, Louie, Ora and Tommy, while the others we will ignore. The method used will be to make, for each character, a convolutional neural network (using the structure above described) which will give a score describing how certain the model is if the face detected is the character's face.

Step 1: Generating the datasets

We will take every face from the Task 1 dataset and check if it is the face of Andy, Louie, Ora or Tommy (we got the label from the input file). If it is the face of one of them, we will add the image crop with the face in the respective character's dataset with the label "0" (as a valid face of the character) and also add it to the other character datasets with the label "1" (as it is not a valid face of the other characters). As you can see, the labels are the exact opposite to the ones used as task 1 (where we used the obvious way of assigning "1" to valid and "0" to invalid), it was not intended at first, but it is now a part of our family!

```
for i in range(len(valid_faces)):
    # If we have a photo of character X in dataset, it will represent a positive example for
    # the respective character classifier and a negative one for the other classifiers
    if valid_faces_character[i] == "andy":
        task2_andy_training_examples.append(valid_faces[i])
        task2_andy_training_labels.append(0)

        task2_louie_training_examples.append(valid_faces[i])
        task2_louie_training_labels.append(1)
        task2_ora_training_examples.append(valid_faces[i])
        task2_ora_training_labels.append(1)
        task2_tommy_training_examples.append(valid_faces[i])
        task2_tommy_training_labels.append(1)
```

Fig 6. The valid faces generation for Andy's recognition dataset

As it can be observed in Figure 6, we append a “0” (valid) in Andy’s recognition dataset and “1” (invalid) in the other character’s dataset. As mentioned above, we do the exact same algorithm for the other characters as well.

Step 2: Training the model

As mentioned at the Task 2 introduction part, we use the same model structure (CNN) that we used at Task 1, except we doubled the learning rate (0.0002) and we now train for just 9 epochs.

```
#For each character we train a classifier that decides whether a photo contains or does not contain that character

#Andy
if from_saved_model_task2:
    model_task2_andy = keras.models.load_model(saved_model_task2_andy_path)
    print("Am incarcat modelul pentru Andy")
else:
    print("Se antreneaza modelul pentru Andy")
    model_task2_andy = keras.models.Sequential([
        keras.layers.Conv2D(filters = 32, kernel_size = (2, 2), activation = 'relu', input_shape = (box_size, box_size, 3)),
        keras.layers.MaxPooling2D(),

        keras.layers.Conv2D(filters = 64, kernel_size = (2, 2), activation = 'relu'),
        keras.layers.MaxPooling2D(),

        keras.layers.Conv2D(filters = 128, kernel_size = (2, 2), activation = 'relu'),
        keras.layers.MaxPooling2D(),

        keras.layers.Flatten(),

        keras.layers.Dense(256, activation = 'relu'),
        keras.layers.Dropout(0.25),

        keras.layers.Dense(64, activation = 'relu'),
        keras.layers.Dropout(0.25),

        keras.layers.Dense(32, activation = 'relu'),
        keras.layers.Dropout(0.25),

        keras.layers.Dense(16, activation = 'relu'),

        keras.layers.Dense(2, activation = 'softmax')
    ])

    opt = keras.optimizers.Adam(learning_rate = 2e-4)
    model_task2_andy.compile(optimizer = opt, loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
    model_task2_andy.fit(task2_andy_training_examples, task2_andy_training_labels, epochs = 9, shuffle = 'True')
```

Fig 7. Andy’s recognition model structure

Step 3: Generating the predictions

Now that we have the classifiers ready, we can predict the detected faces. For all detections found at task 1, we will predict with each classifier if the face is of the respective character or not by considering any face that obtains at least 0.9 score (out of 1) as a valid face of the character who's classifier gave the score (if Tommy's classifier gives 0.91 on a face, we label it as Tommy's face).

```
faces = np.array(faces)

#Predict on the list of faces(scores between 0 and 1 if in the photo is andy/louie/ora/tommy)
andy_predictions = model_task2_andy.predict(faces)
louie_predictions = model_task2_louie.predict(faces)
ora_predictions = model_task2_ora.predict(faces)
tommy_predictions = model_task2_tommy.predict(faces)

#Score threshold to consider that the photo contains the face of a character
threshold = 0.9
for i in range(len(andy_predictions)):

    #Debug info
    if i % 100 == 0:
        print(f"Suntem la detectia {i}: poza {file_names[i]} coordonatele {detections[i]}")

    img = cv.imread(f"{path}/{file_names[i]}")
    img = img[int(detections[i][1]) : int(detections[i][3]), int(detections[i][0]) : int(detections[i][2])]

    #Checking the score for each character if it is within the required threshold
    if andy_predictions[i][0] >= threshold:
        andy_detections.append(detections[i])
        andy_scores.append(andy_predictions[i][0])
        andy_file_names.append(file_names[i])

    if louie_predictions[i][0] >= threshold:
        louie_detections.append(detections[i])
        louie_scores.append(louie_predictions[i][0])
        louie_file_names.append(file_names[i])

    if ora_predictions[i][0] >= threshold:
        ora_detections.append(detections[i])
        ora_scores.append(ora_predictions[i][0])
        ora_file_names.append(file_names[i])

    if tommy_predictions[i][0] >= threshold:
        tommy_detections.append(detections[i])
        tommy_scores.append(tommy_predictions[i][0])
        tommy_file_names.append(file_names[i])
```

Fig 8. The algorithm described above

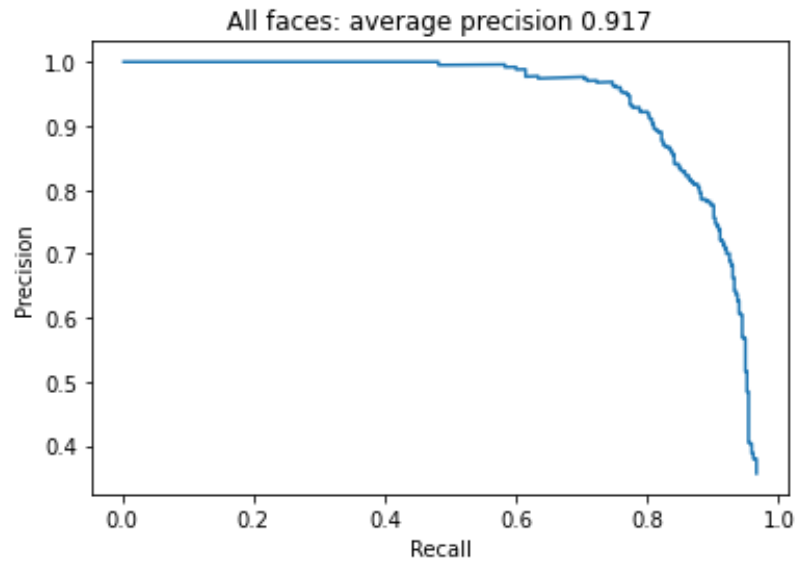


Fig 9. The score obtained by the algorithm at Task 1 on validation

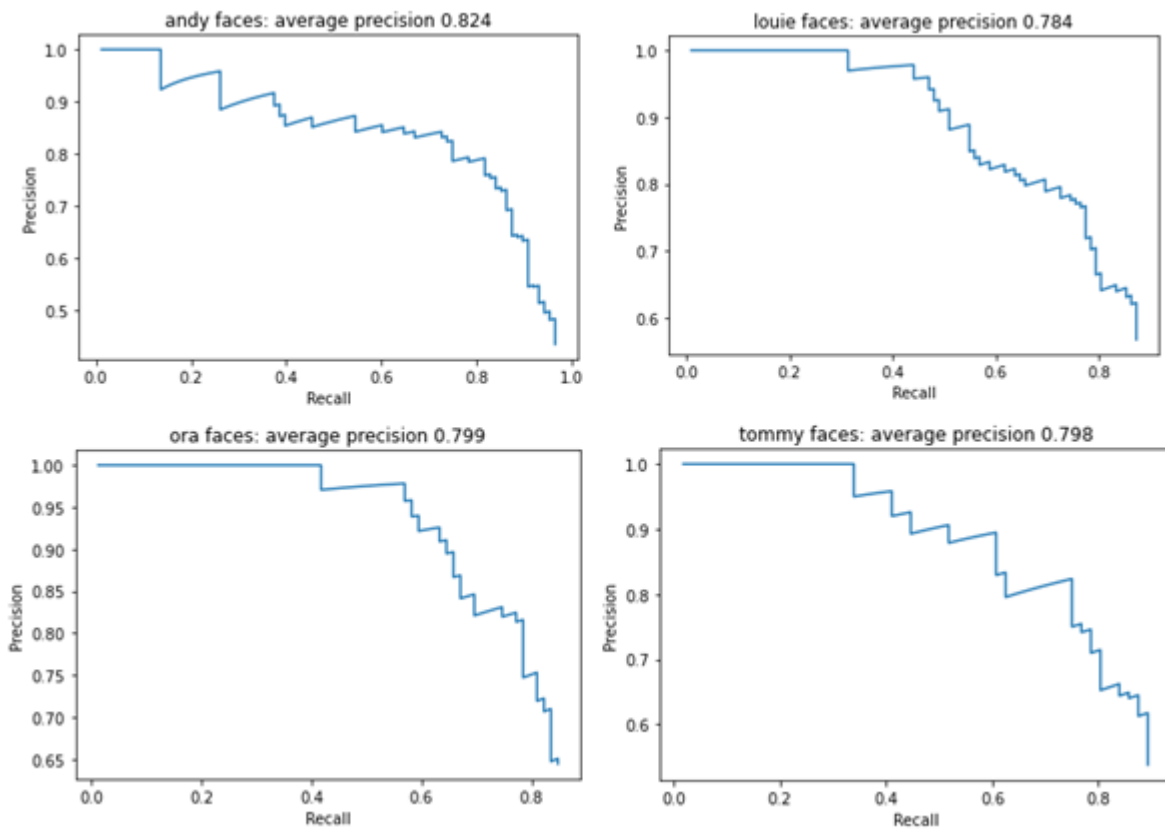


Fig 10. Precisions obtained by the algorithm at Task 2 on validation (median average precision is 0.801)