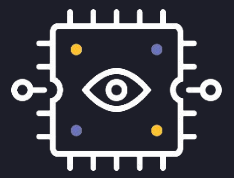


MACHINES
CAN SEE

Video Face Recognition Challenge

MCS2019

April 29 - June 22

Alexander Parkin

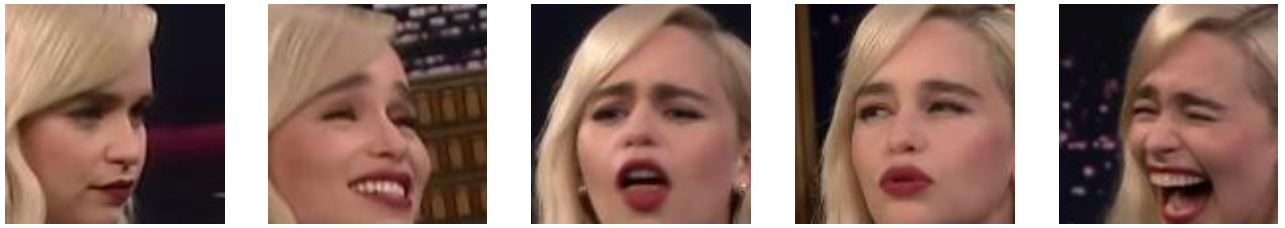
VisionLabs

Introduction

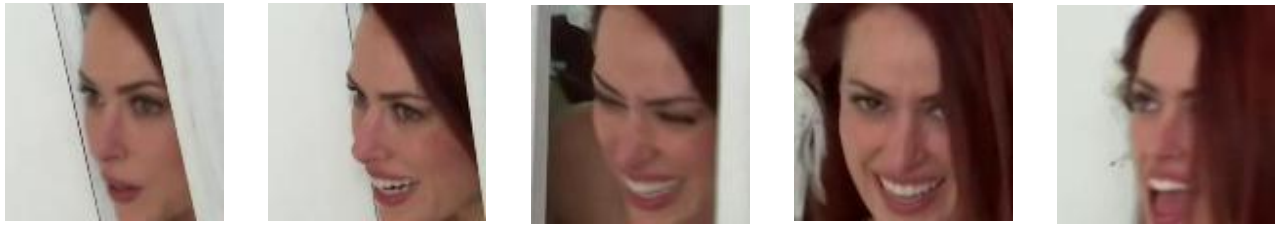
Difficult cases for video face recognition

Motion blur

Facial expressions



Overlaps



Head rotation

Video Face Recognition

→ Face recognition CNN → Descriptor

→ Face recognition CNN → Descriptor, Descriptor, Descriptor

Video Face Recognition

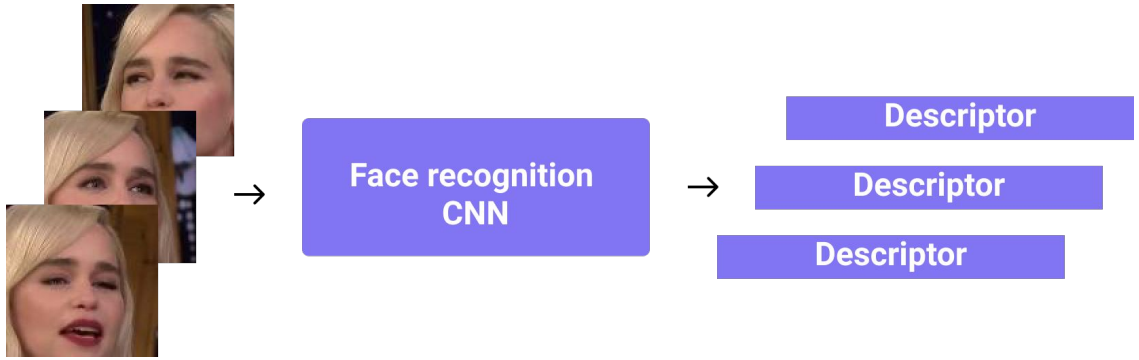
→ Face recognition CNN → Descriptor

→ Face recognition CNN → Descriptor, Descriptor, Descriptor

How to aggregate?

Baseline

1. Extract descriptors



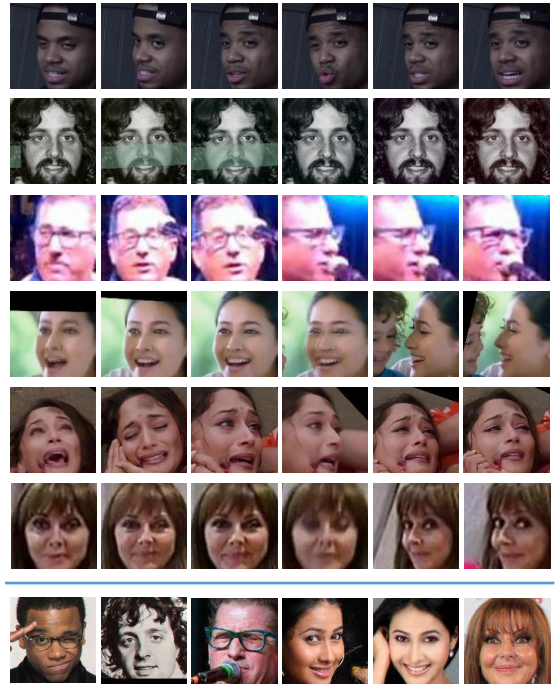
2. Aggregation

$$\frac{1}{10}\sum_{i=1}^{10}\left(\text{i-th frame face descriptor}\right)$$

3. Track descriptor normalization

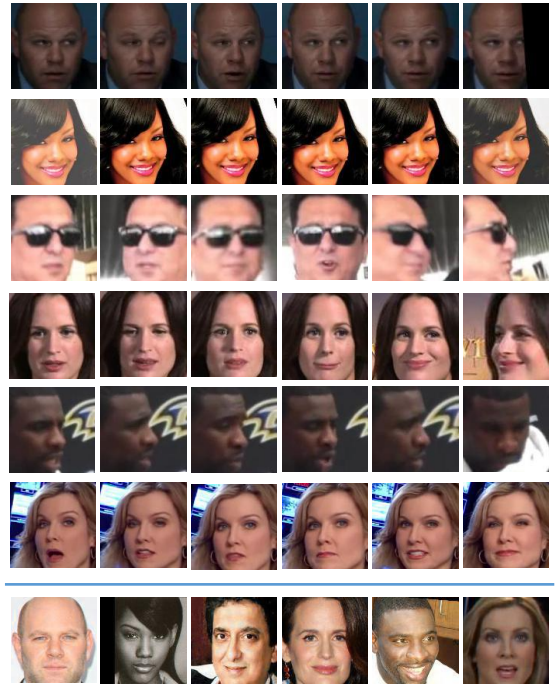
MCS Challenge dataset

Train



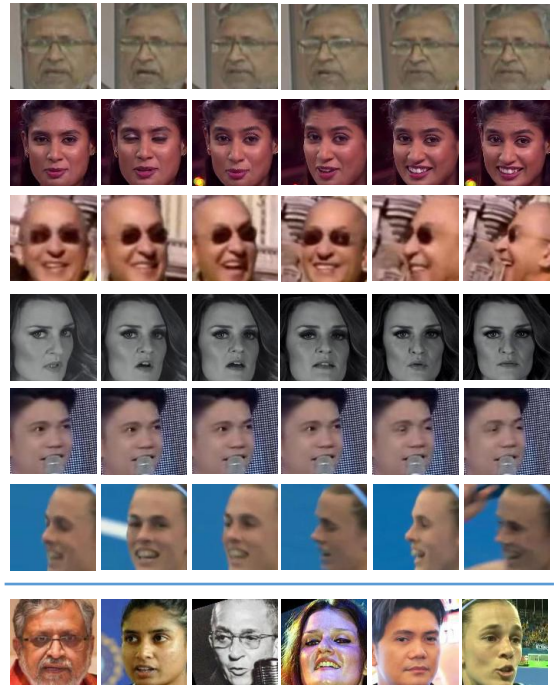
Tracks: 15 534
Identities: 935

Public test



Tracks: 4025
Identities: 275

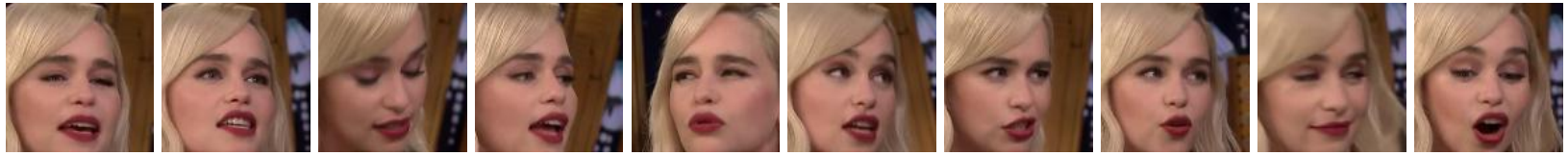
Private test



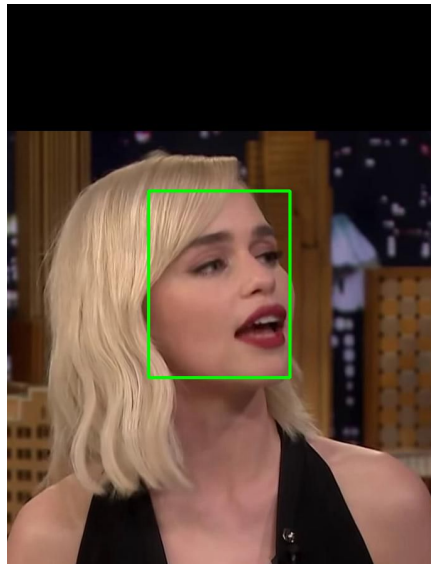
Tracks: 4025
Identities: 4025

Data description

Track



Face crop

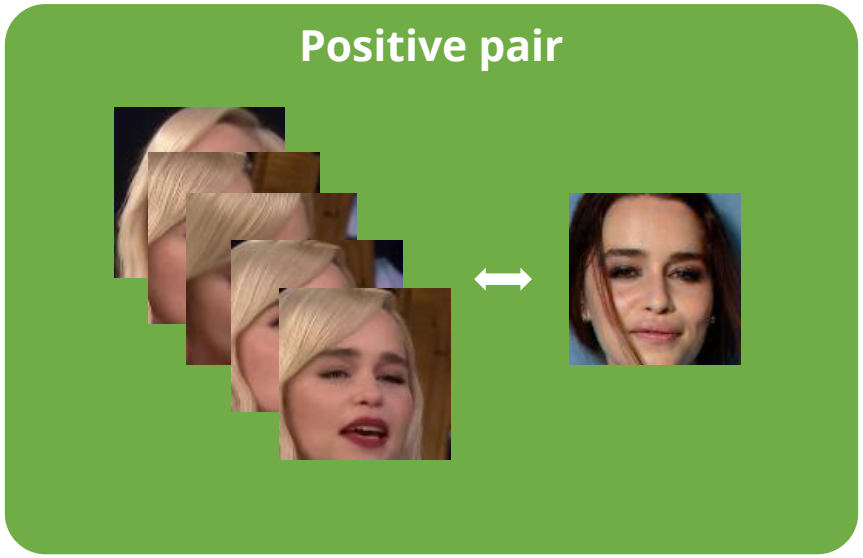


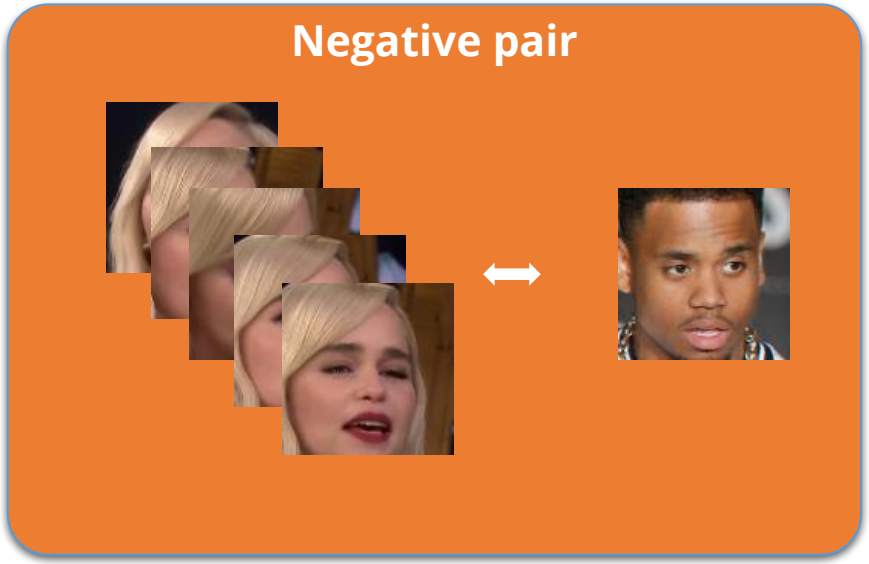
Face bbox

5 landmark points

Aligned face warp

Test scoring





TPR@FPR =1e-5

DevBattle platform

1. Participant’s solutions must be in the same conditions
2. Participant’s solutions must be a bit like production

OS: Ubuntu 18.04
CPU: i7-8700 (6 cores)
RAM: 16 Gb
GPU: 2080Ti
Execution time: 15 min

https://devbattle.ru

Gitlab CI/CD platform

Results

#	Team	Public TPR@FPR10e-6	Private TPR@FPR10e-6
1	2a5466170931aa020ad86ebcd35d76d3	0.8790	**0.8932**
2	Toohardforus	0.8606	**0.8787**
3	Test team	0.7027	**0.8571**
4	So wow team	0.8915	**0.8298**
5	Voronezh-city-of-goD	0.8104	**0.8257**
6	gc	0.7878	**0.8166**
7	griddic	0.8127	**0.8151**
8	bragin	0.7611	**0.8116**
9	BearStrikesBack	0.7474	**0.7851**
10	Blindless	0.6886	**0.7310**
11	gooR0sho	0.7045	**0.7278**
12	PIVO_NA_POLU	0.6968	**0.7267**
13	figak figak and in production	0.6991	**0.7211**
14	stadyo	0.6527	**0.7040**
16	Baseline	0.6514	**0.7039**

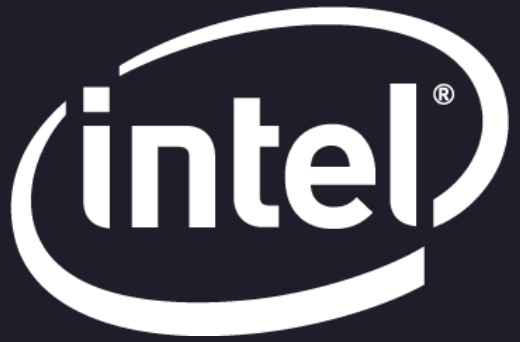
53 registered teams -> 14 above baseline

Prizes

1. **250 000 RUB** + Intel NUC L10 (CPU: i7)
   + depth camera Intel® RealSense™ D415
   + Intel® Neural Compute Stick 2

2. **150 000 RUB** + Intel NUC L10 (CPU: i5)
   + depth camera Intel® RealSense™ D415
   + Intel® Neural Compute Stick 2

3. **100 000 RUB** + Intel NUC L10 (CPU: i5)
   + depth camera Intel® RealSense™ D415
   + Intel® Neural Compute Stick 2



Prizes

1. **250 000 RUB** + Intel NUC L10 (CPU: i7)
   + depth camera Intel® RealSense™ D415
   + Intel® Neural Compute Stick 2

2. **150 000 RUB** + Intel NUC L10 (CPU: i5)
   + depth camera Intel® RealSense™ D415
   + Intel® Neural Compute Stick 2

3. **100 000 RUB** + Intel NUC L10 (CPU: i5)
   + depth camera Intel® RealSense™ D415
   + Intel® Neural Compute Stick 2

Prizes

1. **250 000 RUB** + Intel NUC L10 (CPU: i7)
   + depth camera Intel® RealSense™ D415
   + Intel® Neural Compute Stick 2

2. **150 000 RUB** + Intel NUC L10 (CPU: i5)
   + depth camera Intel® RealSense™ D415
   + Intel® Neural Compute Stick 2

3. **100 000 RUB** + Intel NUC L10 (CPU: i5)
   + depth camera Intel® RealSense™ D415
   + Intel® Neural Compute Stick 2

Alexander Kiselev,
Tatiana Gaintseva

Prizes

1. **250 000 RUB** + Intel NUC L10 (CPU: i7)
   + depth camera Intel® RealSense™ D415
   + Intel® Neural Compute Stick 2

   Mamat Shamshiev, Artem Tsypin, Vladislav Filimonov

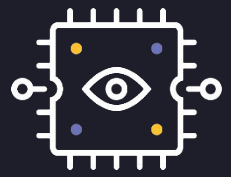
2. **150 000 RUB** + Intel NUC L10 (CPU: i5)
   + depth camera Intel® RealSense™ D415
   + Intel® Neural Compute Stick 2

3. **100 000 RUB** + Intel NUC L10 (CPU: i5)
   + depth camera Intel® RealSense™ D415
   + Intel® Neural Compute Stick 2

   Alexander Kiselev, Tatiana Gaintseva

Prizes

1.	250 000 RUB + Intel NUC L10 (CPU: i7) + depth camera Intel® RealSense™ D415 + Intel® Neural Compute Stick 2	Vasily Solodovnikov
2.	150 000 RUB + Intel NUC L10 (CPU: i5) + depth camera Intel® RealSense™ D415 + Intel® Neural Compute Stick 2	Mamat Shamshiev, Artem Tsypin, Vladislav Filimonov
3.	100 000 RUB + Intel NUC L10 (CPU: i5) + depth camera Intel® RealSense™ D415 + Intel® Neural Compute Stick 2	Alexander Kiselev, Tatiana Gaintseva



MACHINES
CAN SEE

MCS2019 competition

1st place solution

Vasily Solodovnikov

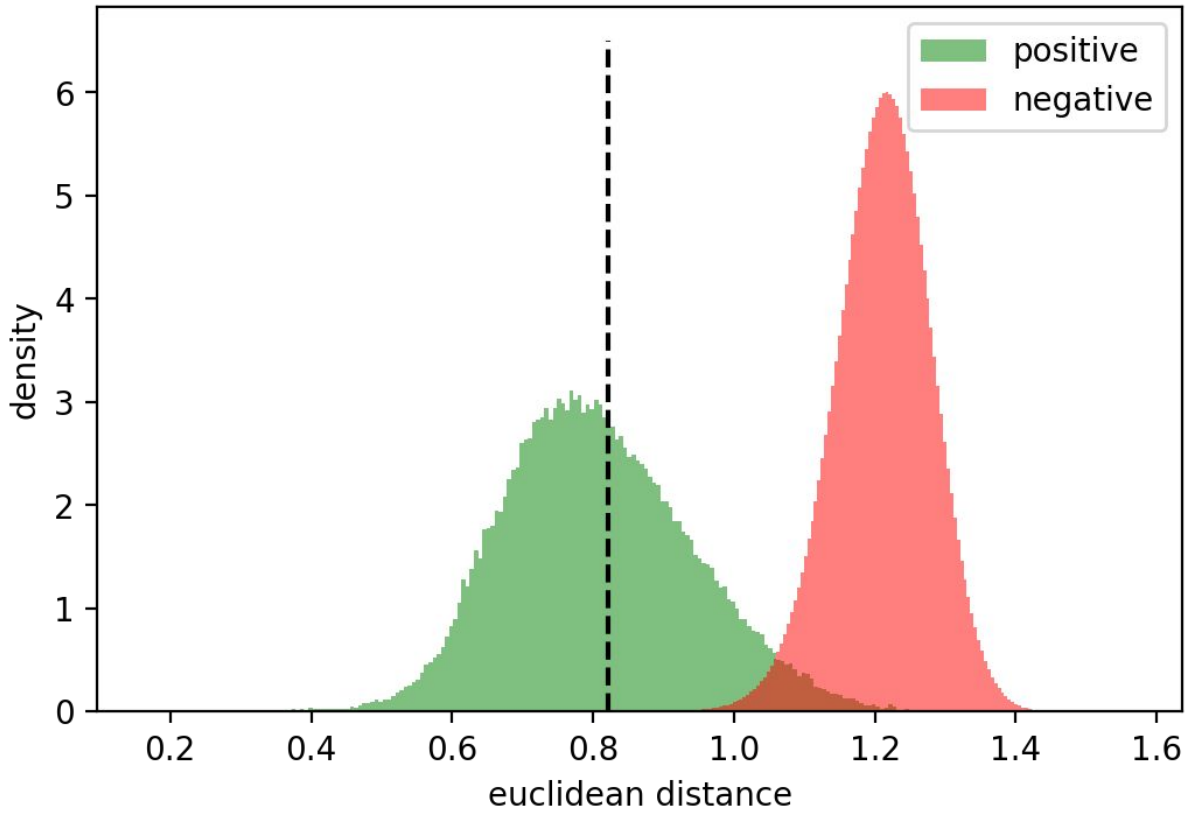
Team “2a5466170931aa020ad86ebcd35d76d3”

Baseline solution

Average out 10 track embeddings and project the result onto unit hypersphere

Threshold at FPR=10^{-5} - 0.82
TPR - 0.568

Can we do better?..



Distribution of pairwise Euclidean distances

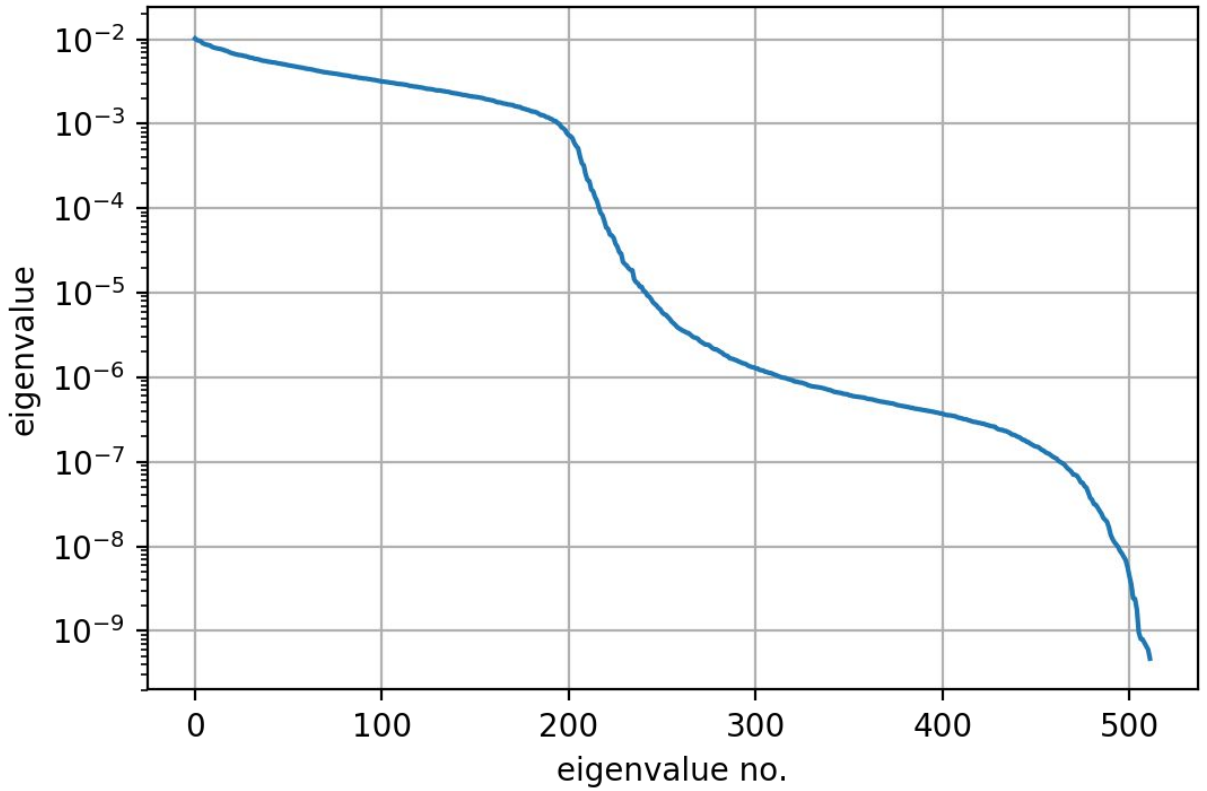
Data analysis

Principal components

Despite the provided embeddings are 512-D, they appear to reside in 200-D subspace

Projecting them to 200-D before feeding to neural network is beneficial for 2 reasons:

- orthogonality
- reduced noise



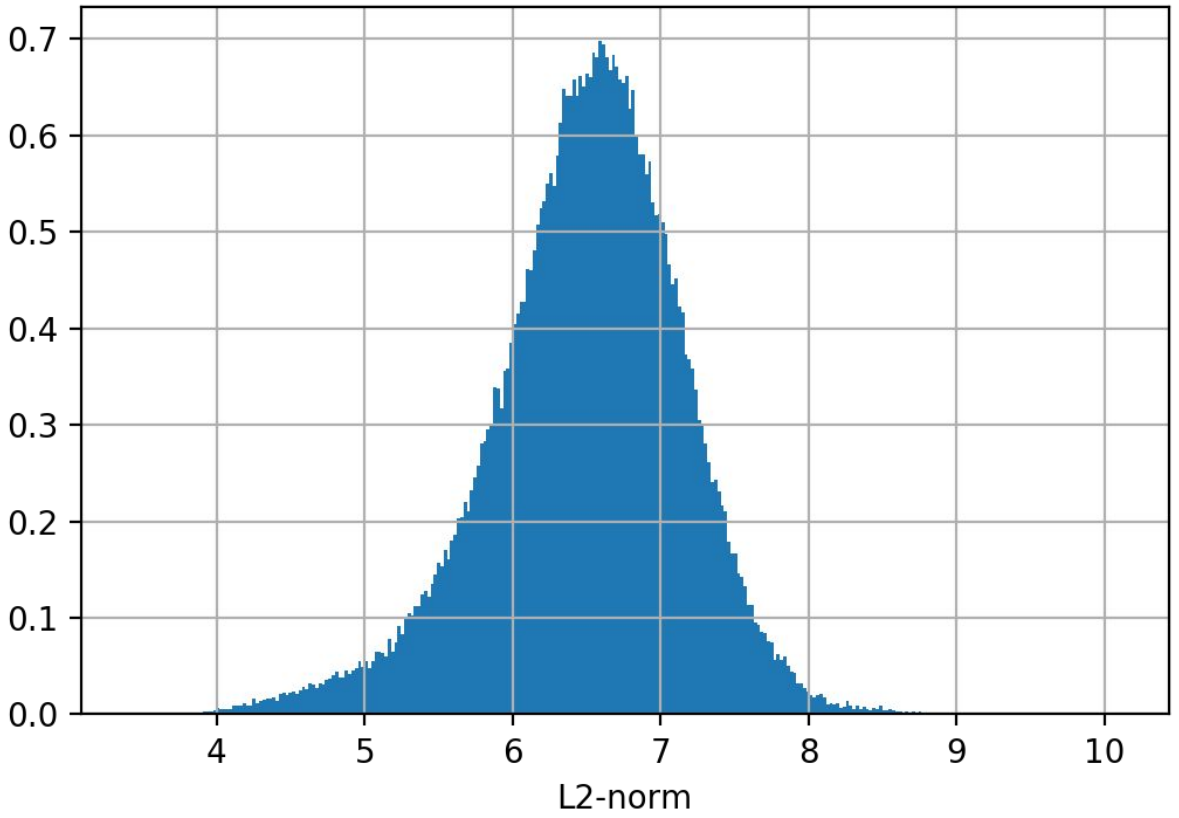
Eigenvalues (log-scale)

Data analysis

Magnitudes of raw embeddings

Provided track embeddings are unit-normalized

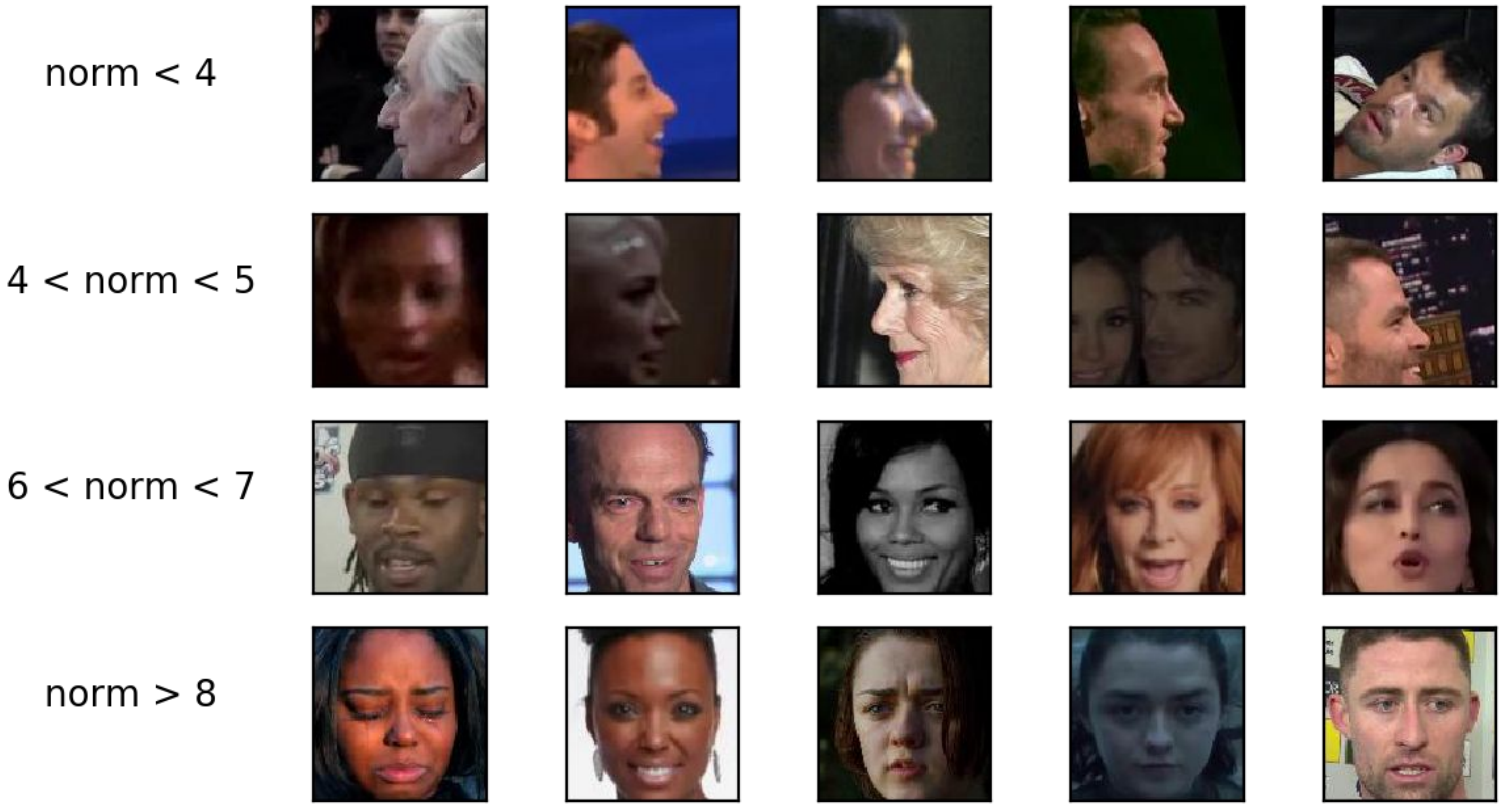
However, one can perform inference with normalization disabled



Distribution of embedding L2-norms

Data analysis

Embedding magnitude as a measure of “uncertainty”

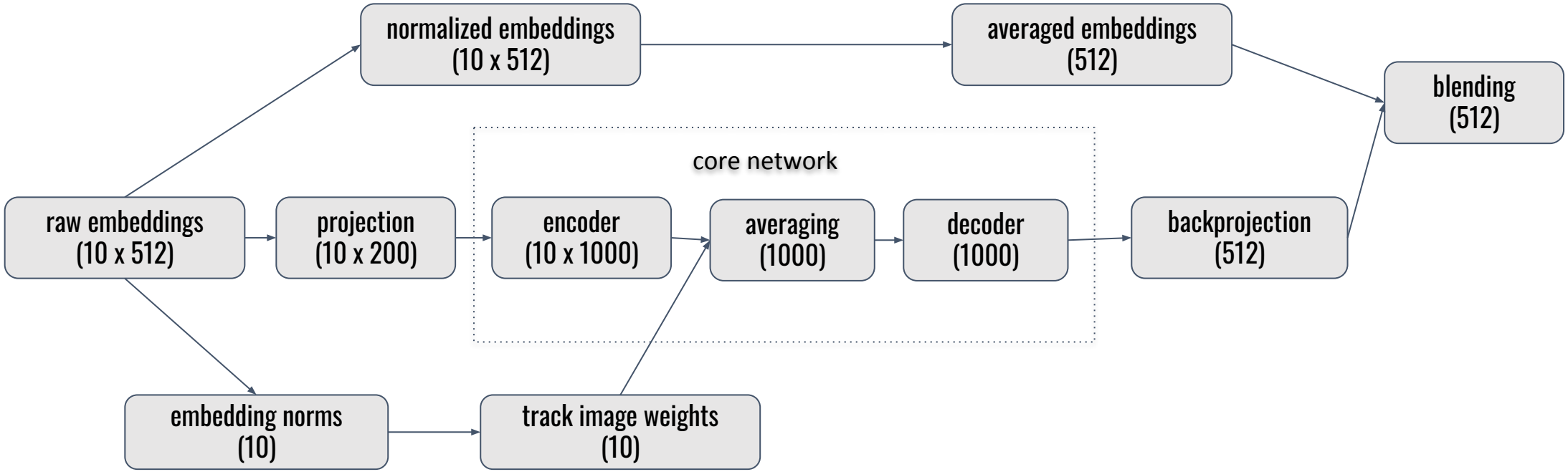


Data analysis

Embedding magnitude as a measure of “uncertainty”

Neural Network

Overall structure



Neural Network

Core

- Applying *softmax* activation is crucial
- Reminiscent of von Mises-Fisher mixture model

```
input = Input([None, 200])

encoder = Dense(1000, bias_initializer=tf.constant_initializer(0.))
decoder = Dense(200, use_bias=False)

tmp = encoder(input)
codes = Softmax(axis=-1)(tmp)
code = GlobalAveragePooling1D()(codes)
output = decoder(code)
```

Training

Distance calculation

- Randomly sample a batch of 128 tracks
- Apply the network to produce aggregated embeddings
- Calculate pairwise Euclidean distances to each of the *ground truth* embeddings. Produced tensor has a shape of 128 x 935 x 5
- In order to avoid high memory consumption, don't calculate Euclidean distances directly. First, use `einsum` to compute cosine similarities, then convert them to Euclidean distances
- Compute 128 x 935 boolean mask to distinguish between positive and negative pairs

Training

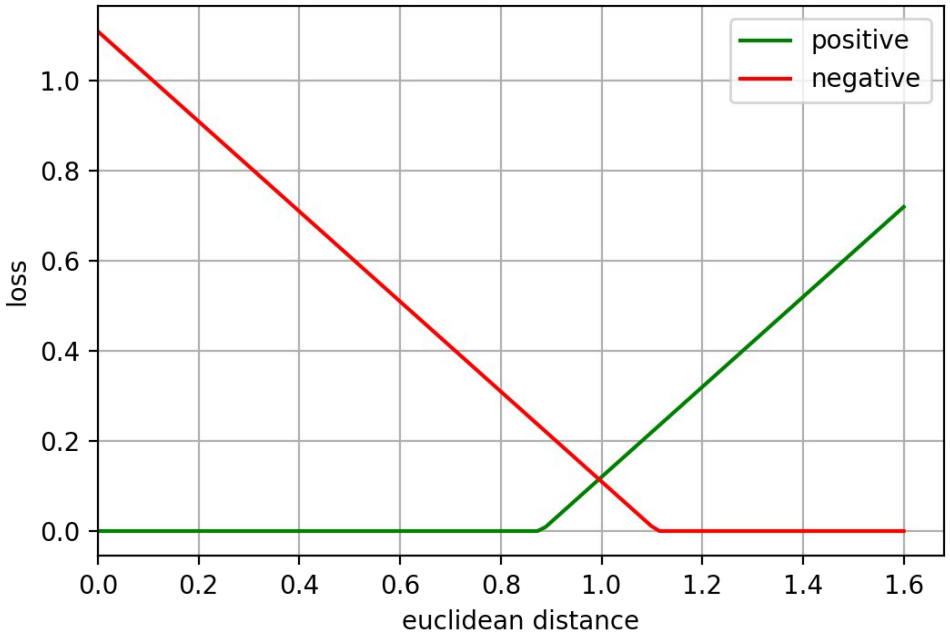
Loss function

Double-margin contrastive loss, e.g.

- *VISALOGY: Answering Visual Analogy Questions* (https://arxiv.org/abs/1510.08973)

```
def calc_loss(distances, is_same):
  dp = tf.maximum(0., tf.boolean_mask(distances, is_same) - config['tp'])
  dn = tf.maximum(0., config['tn'] - tf.boolean_mask(distances, ~is_same))
  return tf.reduce_mean(dp) + config['wn'] * tf.reduce_mean(dn)

config = dict(tp=0.89, tn=1.11, wn=250)
```



Triplet loss also performs well, but converges slower

- *In Defense of the Triplet Loss for Person Re-Identification* (https://arxiv.org/abs/1703.0773)

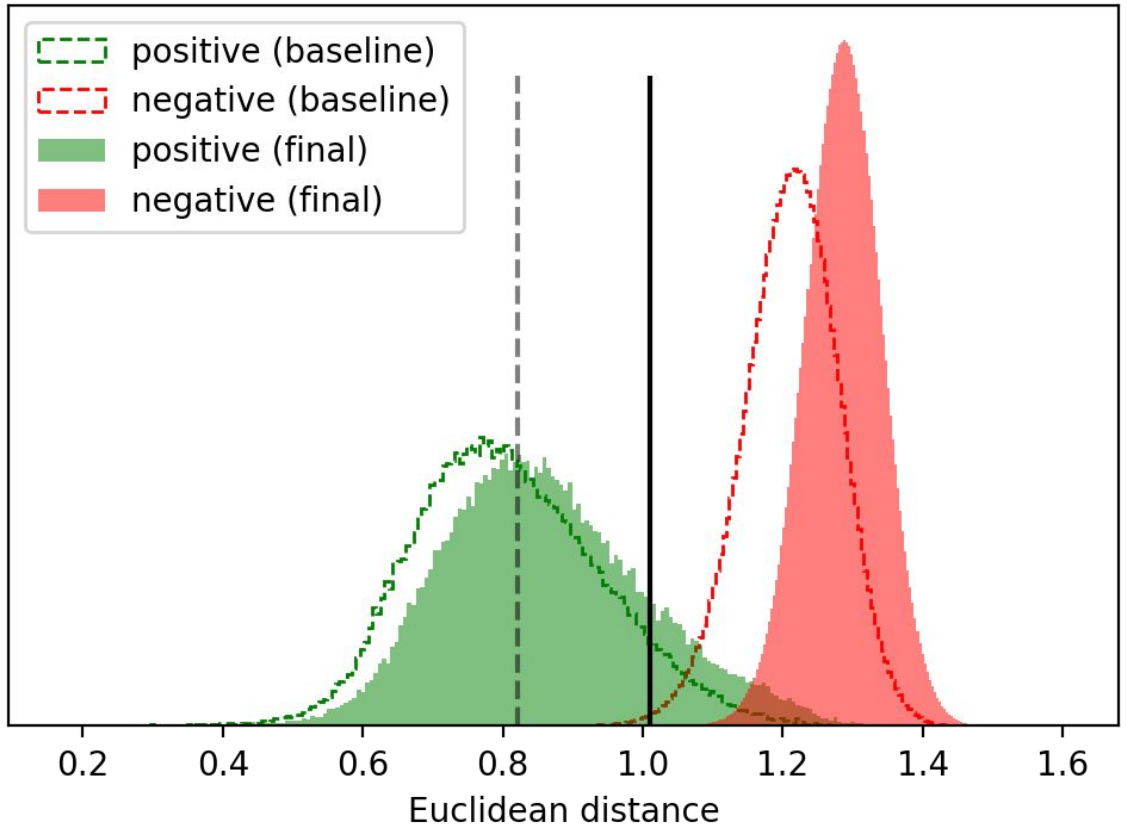
Training

Optimization details

- Adam optimizer, learning rate = $3*10^{-4}$ (reduced to $1*10^{-4}$ after 10 epochs)
- Initial batch size - 128, increase up to 512 after 10 epochs and further up to 1024 after 20 epochs
- Converges in less than 10 minutes on a single-core CPU

Final solution

Model	Mean distance	Threshold	TPR
Baseline	0.806	0.820	0.568
Final	0.861	1.01	0.844



Distributions of pairwise Euclidean distances

Additional tricks

- Use LR-flips for test time augmentation (+0.5%)

Video Face Recognition Challenge

Artem Tsypin, Mamat Shamshiev,
Vladislav Filimonov

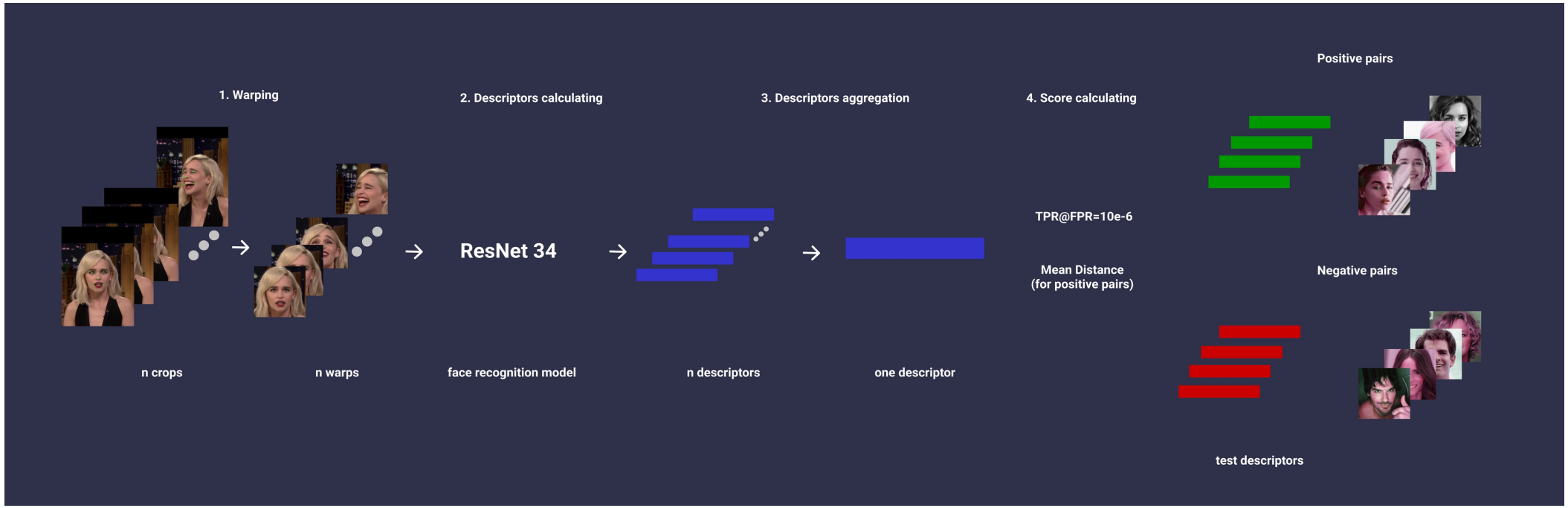
Team Toohardforus

Department of Mathematical Forecasting Methods
CMC MSU

Introduction

Problem statement

Aggregate video stream descriptors provided by a pretrained single-frame CNN. Designed model should minimize the distance between aggregated descriptor and ground truth descriptors and maximize the distance to all other descriptors.



Pic.1 Pipeline.

Data augmentation

Due to hardware limitations only provided descriptors were considered in our solution. Raw images were not used for training.
To reduce overfitting following augmentations of track descriptors were used:

Pic.1 Normal track

Pic.2 Reversed track

Pic.3 Permuted track

Model architecture

Complex models with non-linearities were overfitting which limited the range of possible architectures to fully connected layers and linear combinations of track descriptors.

Final model turned out to be three fully connected layers (one for each track type) with two levels of attention. Average was added to model's output as it is a good initial evaluation.

$$X_{track} = \{x_1, \ldots, x_{10}\}, X_{track} \in \mathbb{R}^{10\times512}, x_i \in \mathbb{R}^{512}.$$

$$\mathbf{Out} = Normalize(\frac{1}{10}\sum_{i=1}^{10} x_i + \sum_{j=1}^{3} \beta_j L_j(X^j)).$$

$$X^1 = X_{track}, X^2 = X_{reverse}, X^3 = X_{perm}.$$

$$L_j(X) = Linear_j(5120, 512)(X) + \sum_{i=1}^{10} \alpha_{ji} x_i,$$

$$\alpha_i = Linear_\alpha(512, 1)(x_i) \in (0; 1),$$

$$\beta_j = Linear_\beta(512, 1)(L(X^j)).$$

Training

Loss function

Cosine embedding loss was used. Margin was set to 0.25. The amount of negative samples per one track was increased to 25. Samples closest to ground truth descriptors were chosen as negative examples.

$$loss(x, y) = \begin{cases} 1 - cos(x, y) & , \quad y = 1 \\ max(0, cos(x, y) - margin) & , \quad y = -1 \end{cases}$$

Optimization

ADAM was overfitting so SGD with momentum was used.

One cycle policy was used for learning rate and momentum adjustment. Learning rate varied from 1e-4 to 1e-3.

Model was trained for 7 epochs.

Results

Results on public leaderboard can be found below.

Model	TPR	Mean dist
One fully connected layer with triplet loss	0.7995	0.8871
One fully connected layer with track permutation and triplet loss	0.8122	0.8729
One fully connected layer with track permutation, cosine and hard negatives	0.8478	0.8833

Model	TPR	Mean dist
Ensemble of one layer models	0.8512	0.8815
Four fully connected layers for different amount of frames with track permutation, cosine loss and hard negatives	0.8617	0.9094
Three fully connected layers with reversed and permuted tracks, cosine loss and hard negatives	**0.8606**	**0.8639**

Questions?

Contacts: Artem Tsypin (tsypinartem9@gmail.com), Mamat Shamshiev (mamatshamshiev@yandex.ru),
Vladislav Filimonov (ifilin45@gmail.com).

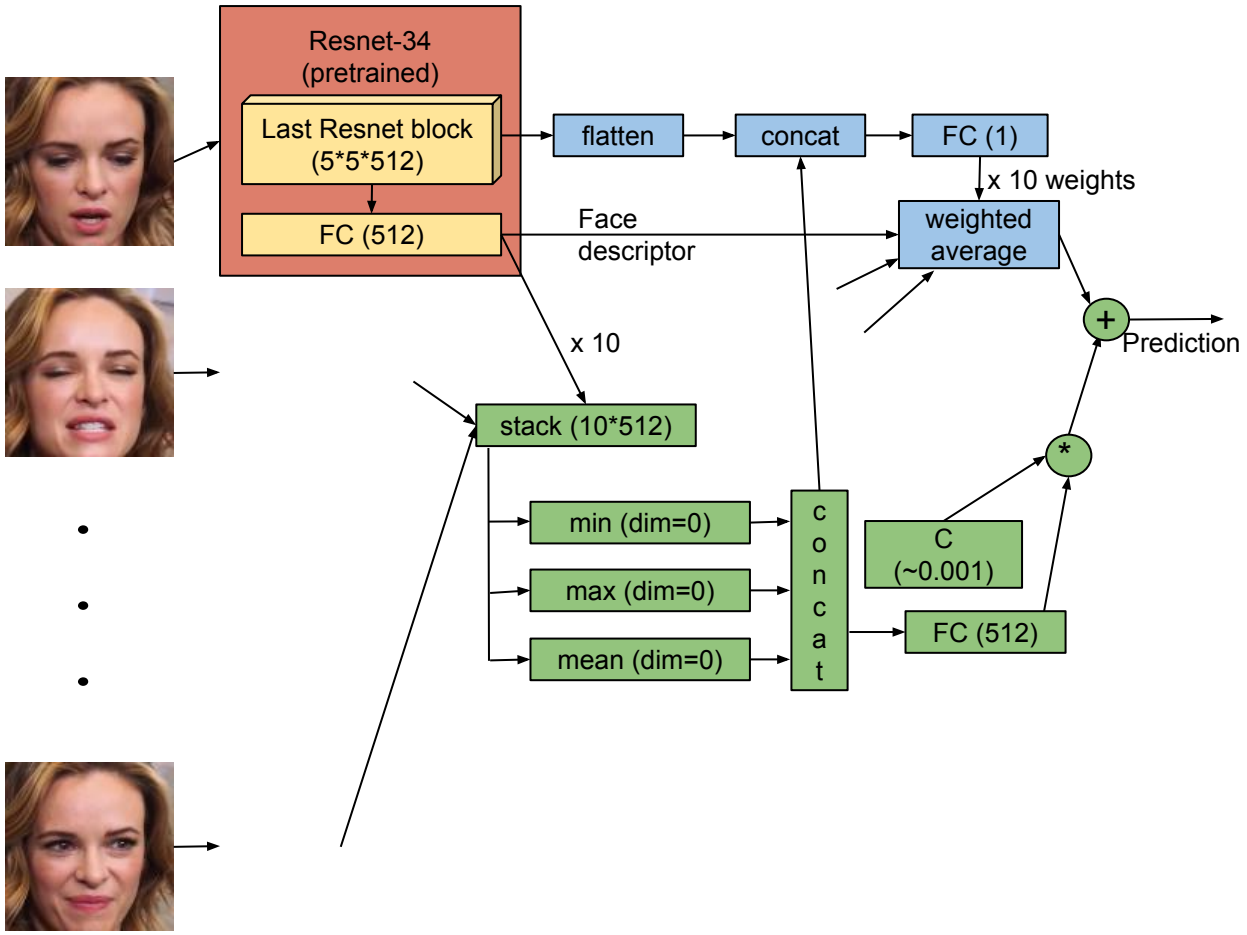
Test Team

Video Face Recognition Challenge

Tatiana Gainseva, Alexander Kiselev

Main idea

1. Calculate descriptors for every image in track
2. Calculate weighted average of these descriptors
3. Add "Correction" vector
4. Train 2 models with different losses and blend their predictions



Network structure

Global features

1. **Extract descriptors for every image in track**
2. **Use symmetric functions to deal with unordered input: mean, min, max**

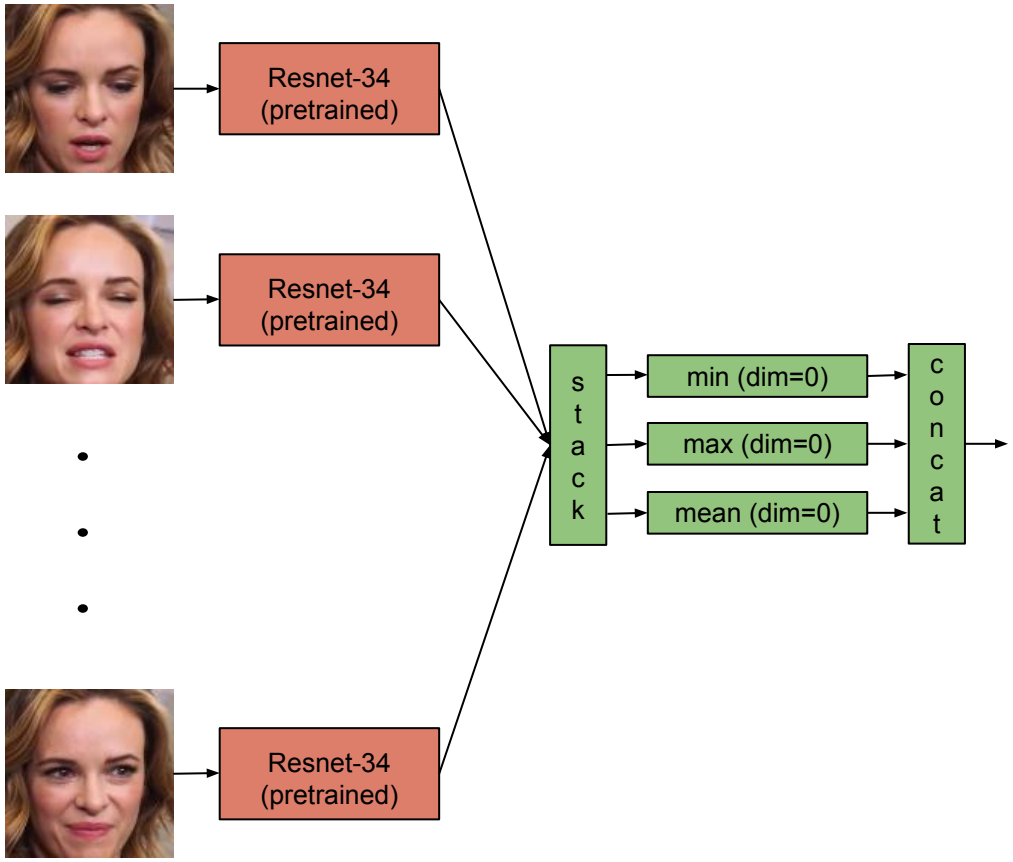
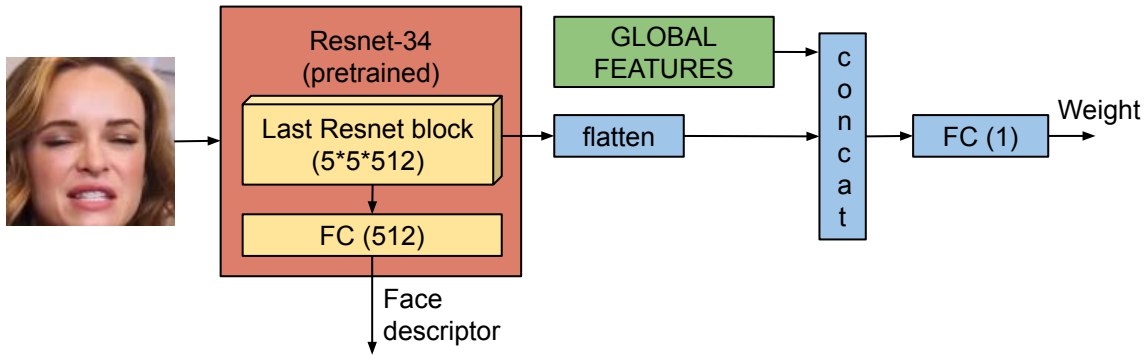


Image weight calculation

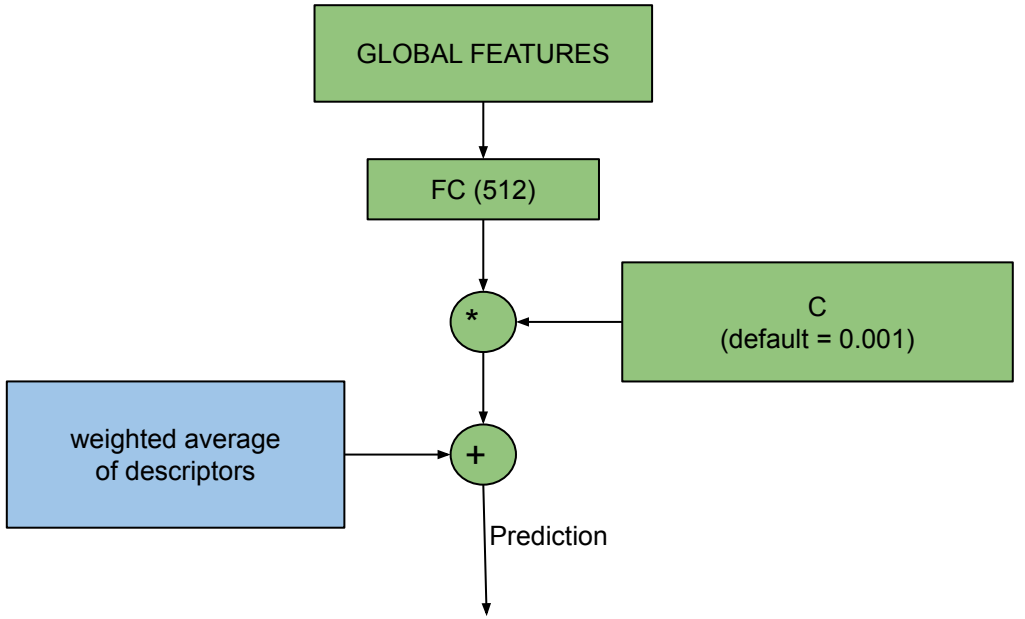
1. "Raw" features from last Resnet block is concatenated with global features
2. Fully connected layer used to calculate weight for current image



Correction vector

1. Calculates from global features via fully connected layer
2. It is added to weighted average of descriptors with coefficient C

C - trainable network parameter with default value of 0.001. That trick helps network at first iterations of training (when correction vector is garbage)



Losses

Model 1:

min(angle(Yp, prediction)) + mean(max(cosine_similarity(Yn, prediction), 0))

Yp - ground truth descriptors of the same person

Yn - mean (per person) descriptors for different persons

Model 2:

min(max(1.0 - cosine_similarity(Yp, prediction), 0)) * (1.0 + sum(max(cosine_similarity(Yn, prediction) - 0.25, 0)))

Yp - ground truth descriptors of the same person

Yn - 2000 random ground truth descriptors of different persons

Ideas summary

What gave more profit:

- **Trying different losses**
- **2 models blending**

What gave less/none profit:

- **"Playing" with network architecture**
- **TTA**
- **Others pretrained models**
- **External datasets**
- **Fine tuning Resnet-34 encoder**

Hacky hacks

Public:

Tried to find same person in predictions and average their descriptors

Private:

4025 tracks of UNIQUE persons...

So we tried to move close descriptors away from each other.
That gave some score, but only on local validation :(

Things that didn’t work

- Applying complicated models which take into account covariances between track embeddings or intra-track affinities
- Using facial keypoints’ coordinates
- Data augmentation by means of small random rotations
- Using *ArcFace* loss. It seems to be a bad proxy for the desired metric and overfits to the training data

Things worth a try

- Filtering out noisy examples from the training data
- More sophisticated way of exploiting embedding magnitudes