Appendix for:
Fast and Interpretable Face Identification for Out-Of-Distribution Data Using Vision Transformers

S1. Pre-trained models

Sources We downloaded the three pre-trained PyTorch models of ArcFace from:

- ArcFace [12]: https://github.com/ronghuaiyang/arcface-pytorch

ArcFace models were trained on dataset CASIA Webface [58].

Architectures The network architectures are provided here:

- ArcFace: https://github.com/ronghuaiyang/arcface-pytorch/blob/master/models/resnet.py

Image-level embeddings for Ranking We use the following layer to extract the image embeddings for stage 1, *i.e.*, ranking images based on the cosine similarity between each pair of (query image, gallery image).

- Arcface: layer bn5 (see code), which is the 512-output, last BatchNorm linear layer of ArcFace (a modified ResNet-18 [24]).

Patch-level embeddings for Re-ranking We use the following layer to extract the spatial feature maps (*i.e.* embeddings $\{q_i\}$) for the patches:

- ArcFace: layer dropout (see code). Spatial dimension: 8×8.

S2. Training hyperparameters

Here we describe the hyperparameters used for ArcFace as follows.

- Margin: $m = 0.5$
- Feature scale: $s = 30.0$

S3. Evaluation metrics

P@1 is well-known as Recall@1 in metric learning. P@1 is computed as follow.

$$\mathcal{N}_q^k = \underset{\mathcal{N} \subset \mathcal{X}_{\text{test}}, |\mathcal{N}|=k}{\arg\min} \sum_{x^f \in \mathcal{N}} d_e\left(\phi\left(x^q\right), \phi\left(x^f\right)\right)$$

where x^q and $\phi(\cdot)$ are inputs and feature encoder respectively, $d_e(\cdot,\cdot)$ is the euclidean distance, and k is k-nearest neighbors. Precision@k can be calculated as:

$$\text{P@}k = \frac{1}{|\mathcal{X}_{\text{test}}|} \sum_{x_q \in \mathcal{X}_{\text{test}}} \frac{1}{k} \sum_{x^i \in \mathcal{N}_q^k} \begin{cases} 1, & y^i = y^q \\ 0, & \text{otherwise} \end{cases}$$

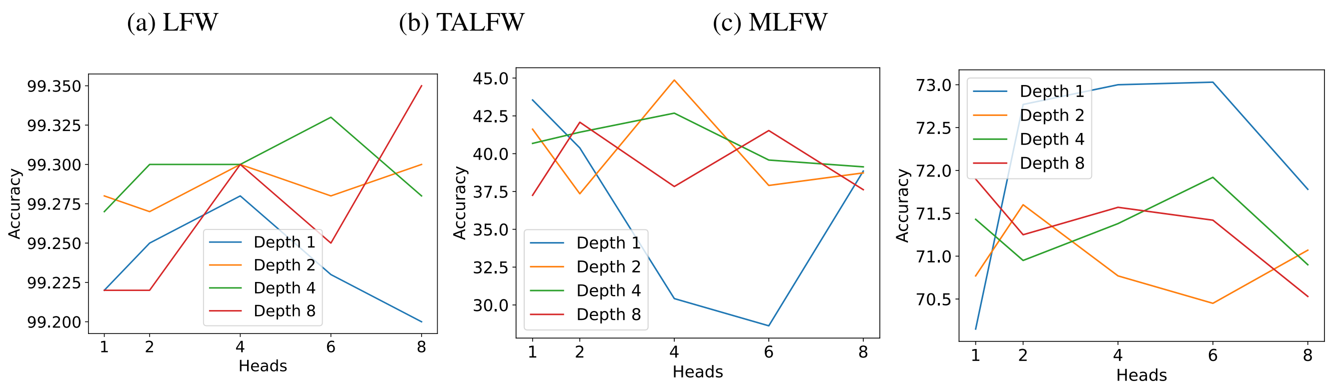


Figure S1. The efficiency of settings of depths and heads for the network (**H2L**) within different domains. For LFW, the depth of 1 achieved comparable accuracy with a depth of 8 (*e.g.* very small difference of 0.075 %). In TALFW, with depths of 1 and 2 and heads of 1 and 4 respectively, the accuracy outperforms the accuracy of depths of 4 and 8. For face masks in MLFW, the depth of 1 consistently outperforms the other settings. Therefore, using a low depth of **1** or **2** for contextual information design can gain good performance.

where y^i is the class label of sample x^i.

To gain more information and a comprehensive ranking evaluation, we computed mean average precision of R (M@R [37]), where R is number of images in a class.

$$M@R = \frac{1}{R}\sum_{i=1}^{R} P(i)$$

where

$$P(i) = \begin{cases} \text{P@}i, & \text{if the } i\text{-th retrieval is correct;} \\ 0, & \text{otherwise.} \end{cases}$$

S4. Time complexity

Here, we evaluate the time complexity of different layers. For vanilla ViT and CNN, the time complexity is mentioned in the Transformer network [51] (see Tab. 5). Hybrid-ViTs consist of convolutional neural networks and low-depth self-attention at the top. Hence, time complexity can be added by convolutional layers and self-attention layers (the last row in Tab. 5).

S5. The low depth's efficiency

For the 2-image/output hybrid-ViT (H2L), adding the Transformer layer at the top of CNN can improve the performance. However, increasing the number of depths and heads can lead to redundant computation in the models while showing no meager improvements in terms of accuracy. Here, we evaluate the effects of different values of depths and heads to select the potential settings in face problems.

Experiment As mentioned above, we use depths $= 1, 2, 4, 8$ and heads $= 1, 2, 4, 6, 8$. We report the accuracy of H2L for face verification on LFW, TALFW, and MLFW datasets.

Results First, we observe that on LFW, the low depth of 1 achieves lower performance. However, it still **outperforms the CNN model** (99.28% Fig. S1 (head=4) vs 98.02% in Fig. 4). Moreover, a lower value of depth and head **achieves comparable results** compared to higher values (*e.g.* 99.22% with depth of 1, head of 1, vs. 99.34% with depth of 8, head of 8). Second, for the TALFW dataset, hybrid-ViTs (H2L) also **achieves comparable accuracy** (with d=1, h=1), *i.e.* performing well with low depth and head values for face adversarial. Third, for the MLFW dataset, the depth of 1 **outperforms** the other higher-depth value models.

S6. User study samples

Fig. S2, Fig. S4, Fig. S4, and Fig. S5 are specific examples for our design for user study. These figures are only the first pages to instruct users for each approach.

Here, we experiment with 4 approaches: no explanation, Hybrid-ViT, CNNs [48], and EMD [40].

S7. Architectures

We provide code snippets for architectures of Transformers, H2L, and H2.

```
class Transformer(nn.Module):
    def __init__(self, dim, depth, heads, dim_head, mlp_dim, dropout):
        super().__init__()
        self.layers = nn.ModuleList([])
        for _ in range(depth):
            self.layers.append(nn.ModuleList([
                Residual(PreNorm(dim, Attention(dim, heads=heads, dim_head=dim_head, dropout=dropout)))
                                                                           ,
                Residual(PreNorm(dim, FeedForward(dim, mlp_dim, dropout=dropout)))
            ]))
    def forward(self, x, mask=None):
        attns = []
        for attn, ff in self.layers:
            att, x = attn(x, mask=mask)
            if att is not None:
                attns.append(att)
            _, x = ff(x)
        return x, attns

class H2L(nn.Module):
    def __init__(self, *, num_class, image_size, patch_size, ac_patch_size,
                     pad, dim, depth, heads, mlp_dim, resnet_model, channels=3,
                     dim_head=64, dropout=0., emb_dropout=0., out_dim=512):
        super().__init__()
        num_patches = 8 ** 2
        patch_dim = channels * ac_patch_size ** 2
        self.resnet_model = resnet_model  # resnet to extract embeedings
        self.sep = nn.Parameter(torch.randn(1, 1, dim))
        self.pos_embedding = nn.Parameter(torch.randn(1, 2*num_patches + 2, dim))
        self.cls_token = nn.Parameter(torch.randn(1, 1, dim))
        self.dropout = nn.Dropout(emb_dropout)
        self.transformer = Transformer(dim, depth, heads, dim_head, mlp_dim, dropout)
        self.to_latent = nn.Identity()
        self.ln = nn.LayerNorm(out_dim)  # Layer norm
        self.soft_split = nn.Unfold(kernel_size=(ac_patch_size, ac_patch_size),
                                    stride=(self.patch_size, self.patch_size),
                                    padding=(pad, pad)
                                    )
        self.patch_to_embedding = nn.Linear(patch_dim, dim)
        self.bn1 = nn.BatchNorm1d(out_dim)
        self.bn2 = nn.BatchNorm1d(out_dim)
        self.fc1 = nn.Linear(d*d*dim, out_dim)
        self.fc2 = nn.Linear(d*d*dim, out_dim)
        self.loss = ArcFace(in_features=out_dim, out_features=num_class)

    def forward(self, img, label=None, mask=None):
        out = self.resnet_model(img)
        x = out['embedding_88']
        N, C, _, _ = x.size()
        x = x.view(N, C, -1).transpose(1, 2)
        b, n, _ = x.shape
        half = int(N/2)

        cls_tokens = repeat(self.cls_token, '() n d -> b n d', b = int(b/2))

        sep = repeat(self.sep, '() n d -> b n d', b=int(b/2))
        splits = torch.split(x, half)
        x = torch.cat((splits[0], sep, splits[1]), dim=1)

        x = torch.cat((cls_tokens, x), dim=1)
        x = self.dropout(x)

        x, attns = self.transformer(x, mask)
        N, d, C = x.size()

        half = int(d/2)
        x1, x2 = x[:, 1:half, :], x[:, (half + 1):d, :]
        embedding1, embedding2 = x1, x2
        f1 = x1.mean(dim=1)
        f2 = x2.mean(dim=1)

        x1 = x1.view(x1.size(0), -1)
        x2 = x2.view(x2.size(0), -1)

        x1 = self.fc1(x1)
        x1 = self.bn1(x1)
        x1 = self.ln(x1)

        x2 = self.fc2(x2)
        x2 = self.bn2(x2)
        x2 = self.ln(x2)

        x = torch.cat((x1, x2), dim=0)
        x = self.loss(x, label)
        return x

class H2(nn.Module):
    def __init__(self, *, num_class, image_size, patch_size, ac_patch_size,
                      pad, dim, depth, heads, mlp_dim, resnet_model, channels=3,
                      dim_head=64, dropout=0., emb_dropout=0., out_dim=512):
        super().__init__()
        num_patches = 8 ** 2
        patch_dim = channels * ac_patch_size ** 2
        self.patch_size = patch_size
        self.resnet_model = resnet_model # # resnet to extract embeedings
        self.sep = nn.Parameter(torch.randn(1, 1, dim))
        self.pos_embedding = nn.Parameter(torch.randn(1, 2*num_patches + 2, dim))
        self.cls_token = nn.Parameter(torch.randn(1, 1, dim))
        self.dropout = nn.Dropout(emb_dropout)
        self.transformer = Transformer(dim, depth, heads, dim_head, mlp_dim, dropout)
        self.to_latent = nn.Identity()
        self.ln = nn.LayerNorm(out_dim)
        self.fc = nn.Linear(dim, 2)  # outputs

    def forward(self, img, label=None, mask=None):
        if self.face_model:
        out = self.resnet_model(img)
        x = out['embedding_88']
        N, C, _, _ = x.size()
        x = x.view(N, C, -1).transpose(1, 2)
        b, n, _ = x.shape
        half = int(N/2)
        cls_tokens = repeat(self.cls_token, '() n d -> b n d', b = int(b/2))
        sep = repeat(self.sep, '() n d -> b n d', b=int(b/2))
        splits = torch.split(x, half)
        x = torch.cat((splits[0], sep, splits[1]), dim=1)
        x = torch.cat((cls_tokens, x), dim=1)
        x += self.pos_embedding[:, :(2*n + 2)]
        x = self.dropout(x)
        x, attns = self.transformer(x, mask)
        N, d, C = x.size()
        x = x[:, 0, :]
        x = self.ln(x)
        x = self.fc(x)
        return x
```

Dataset	Model	# of queries	Time (seconds)	Depth	Head	P@1	RP	M@R
CALFW (Mask)	D DeepFaceEMD [40]	11,914	53.35	-	-	**99.79**	**56.77**	**55.75**
	H2L Hybrid-ViT		**24.33**	1	2	99.29	51.00	50.01
CALFW (Sunglass)	D DeepFaceEMD [40]	12,173	73.90	-	-	**54.95**	30.66	27.74
	H2L Hybrid-ViT		**29.10**	1	6	54.00	**31.00**	**27.87**
AgeDB (Mask)	D DeepFaceEMD [40]	15,629	72.42	-	-	**99.84**	**39.22**	**33.18**
	H2L Hybrid-ViT		**34.44**	1	1	99.28	33.93	26.69
AgeDB (Sunglass)	D DeepFaceEMD [40]	16,409	90.40	-	-	**87.06**	50.04	44.27
	H2L Hybrid-ViT		**33.01**	1	2	86.75	**51.16**	**44.88**

Table S1. Actual running times and performance for ST2 computation in face identification under occlusion. Compared to DeepFace-EMD (D), the computation of hybrid-ViTs (H2L) is significantly faster. For example, for 11,914 query images of the CALFW (mask), H2L runs at least **2** times faster.

Welcome to our user study for face matching!! User study evaluates on domains of masked, sunglass , and normal faces. To make Yes/No decisions, please look at both faces AND the middle visualizations, which highlight the key similarities between the two faces. Try your best to verify matching facial pairs. To answer the question, you can highlight your answers: Yes/No. Use 1st page as the examples. Start in the 2nd page.

Are these two faces of the same person? Your answer: Yes / No

Are these two faces of the same person? Your answer: Yes / No

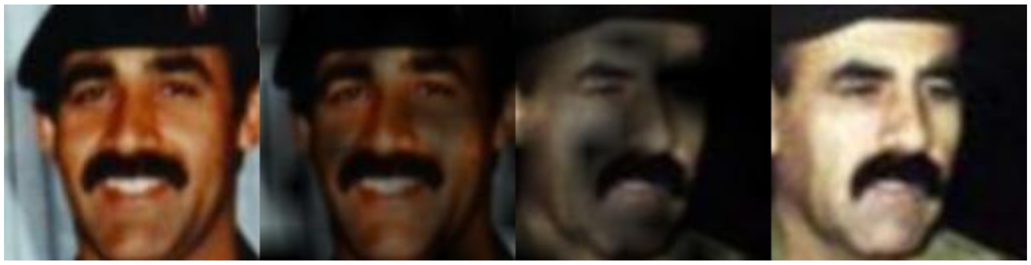
Are these two faces of the same person? Your answer: Yes / No

Are these two faces of the same person? Your answer: Yes / No

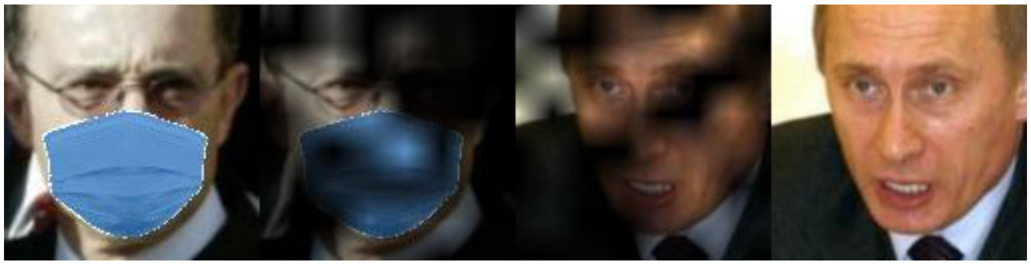
Are these two faces of the same person? Your answer: Yes / No

Figure S2. User study for no-explanation method.

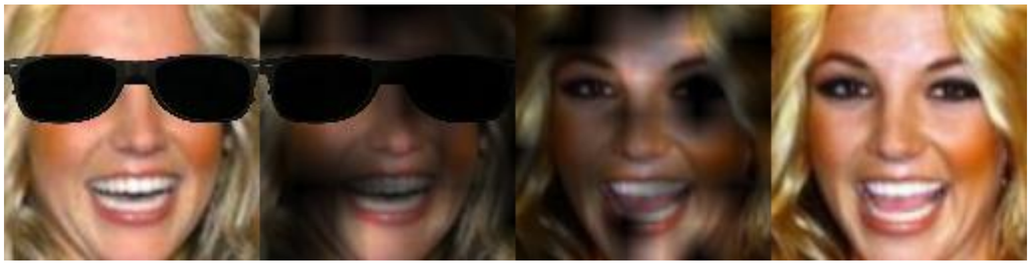
Welcome to our user study for face matching!! User study evaluates on domains of masked, sunglass , and normal faces. To make Yes/No decisions, please look at both faces AND the middle visualizations, which highlight the key similarities between the two faces. Try your best to verify matching facial pairs. To answer the question, you can highlight your answers: Yes/No. Use 1st page as the examples. Start in the 2nd page.



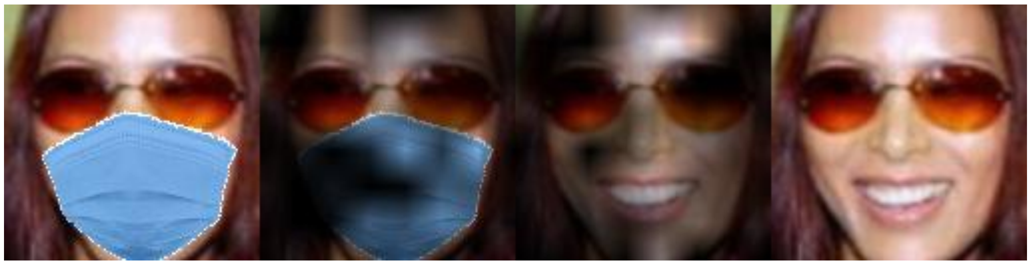
Are these two faces of the same person? Your answer: Yes / No



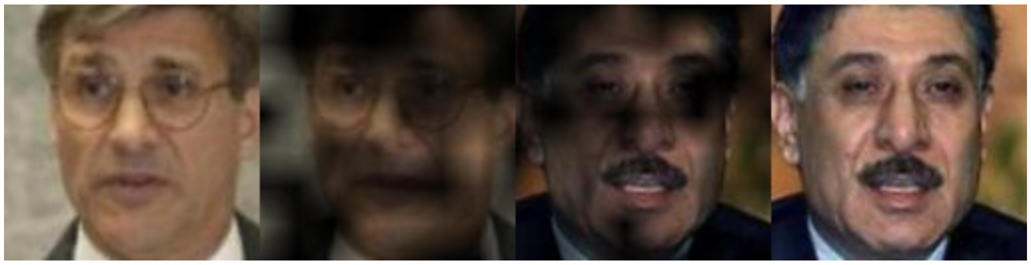
Are these two faces of the same person? Your answer: Yes / No



Are these two faces of the same person? Your answer: Yes / No



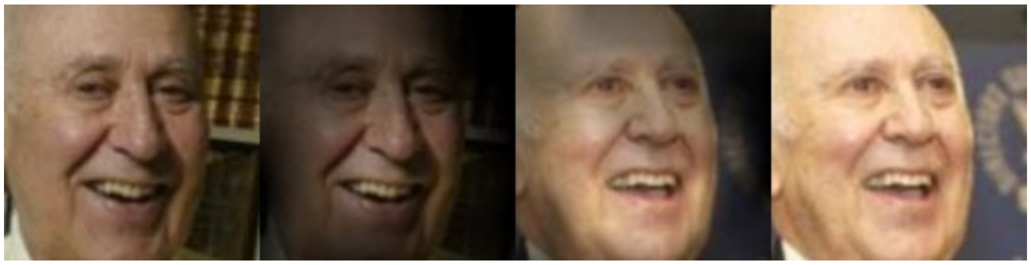
Are these two faces of the same person? Your answer: Yes / No



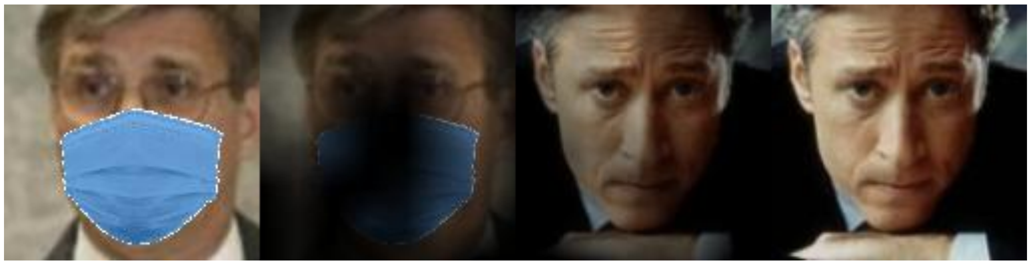
Are these two faces of the same person? Your answer: Yes / No

Figure S3. User study for Hybrid-ViT method.

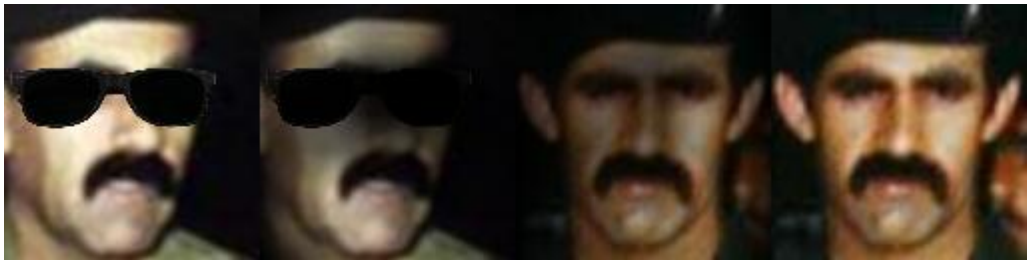
Welcome to our user study for face matching!! User study evaluates on domains of masked, sunglass
, and normal faces. To make Yes/No decisions, please look at both faces AND the middle visualizations, which highlight the key similarities between the two faces. Try your best to verify matching facial pairs. To answer the question, you can highlight your answers: Yes/No. Use 1st page as the examples. Start in the 2nd page.



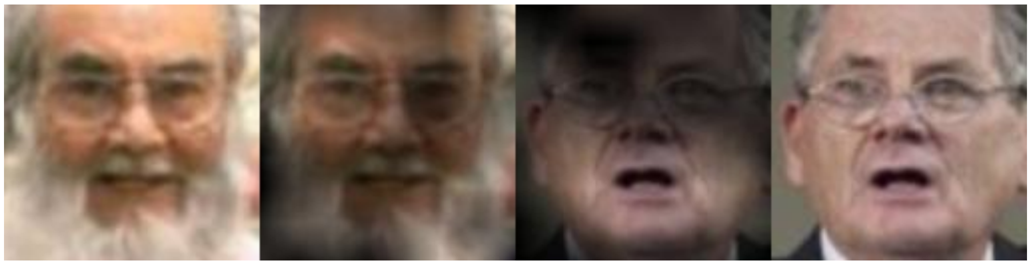
Are these two faces of the same person? Your answer: Yes / No



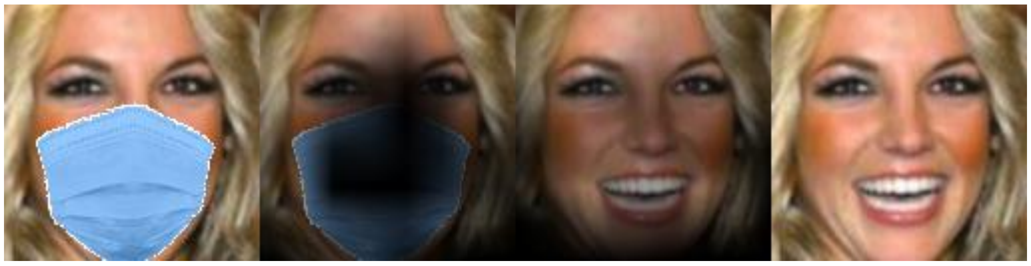
Are these two faces of the same person? Your answer: Yes / No



Are these two faces of the same person? Your answer: Yes / No



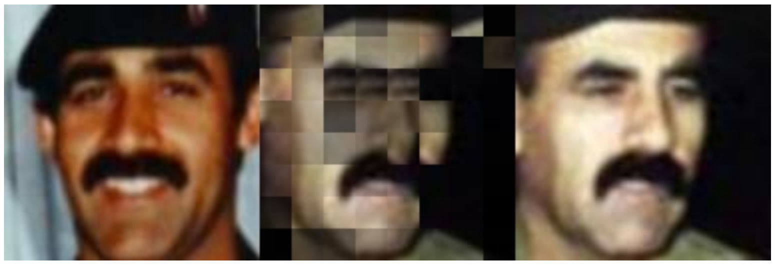
Are these two faces of the same person? Your answer: Yes / No



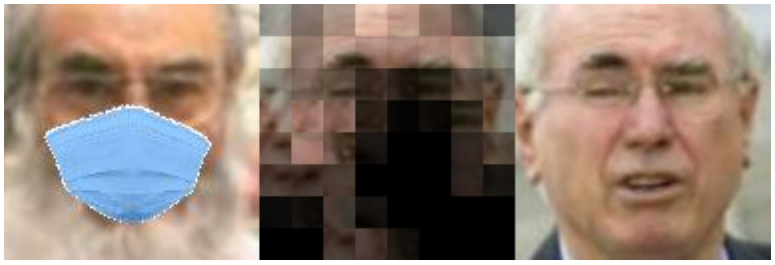
Are these two faces of the same person? Your answer: Yes / No

Figure S4. User study for CNNs method.

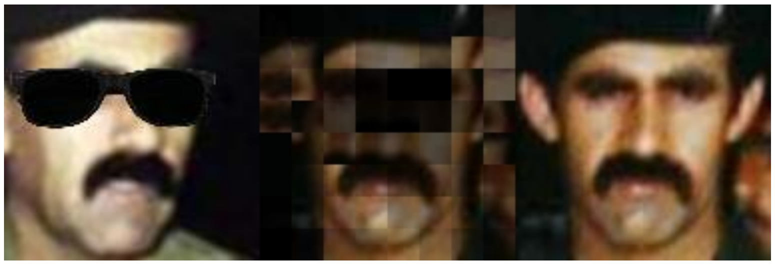
Welcome to our user study for face matching!! User study evaluates on domains of masked, sunglass
, and normal faces. To make Yes/No decisions, please look at both faces AND the middle visualizations, which highlight the key similarities between the two faces. Try your best to verify matching facial pairs. To answer the question, you can highlight your answers: Yes/No. Use 1st page as the examples. Start in the 2nd page.



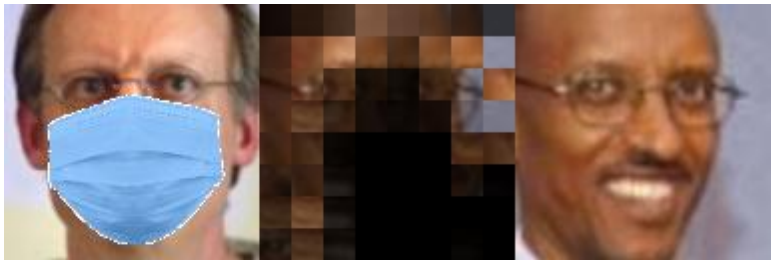
Are these two faces of the same person? Your answer: Yes / No



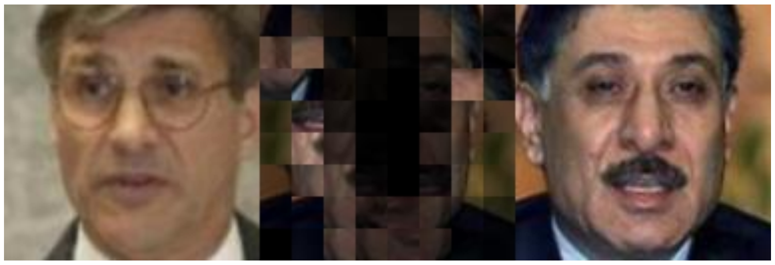
Are these two faces of the same person? Your answer: Yes / No



Are these two faces of the same person? Your answer: Yes / No



Are these two faces of the same person? Your answer: Yes / No



Are these two faces of the same person? Your answer: Yes / No

Figure S5. User study for the EMD method.