Hotel Identification to Combat Human Trafficking

Tom Aarsen
Radboud University, Netherlands
tom.aarsen@ru.nl

Tijn Berns
Radboud University, Netherlands
tijnberns.berns@ru.nl

1 INTRODUCTION

This report includes a detailed description of our 7th place entry on the Hotel-ID to Combat Human Trafficking 2022 Kaggle challenge[1] with a 0.568 MAP@5 score. The goal of this challenge is to identify hotels given a partially obfuscated picture taken from that hotel room. In practice, such a model would be able to assist law enforcement in tracking down victims of human trafficking.

This is the second iteration of this challenge, following the Hotel-ID to Combat Human Trafficking 2021 Kaggle challenge[2]. In contrast to last year's challenge, this new challenge includes obfuscation, allowing law enforcement to use the models on images in which the victims are obfuscated. Furthermore, this new challenge consists of 44.703 training images from 3.116 hotels, as opposed to last year's 97.556 training images from 7.770 hotels.

The report is structured as follows: First, we describe the dataset and different augmentation techniques that are considered. Second, we describe the different models of our training method, as well as our training process. Next, we describe the experiments we conducted and the results we obtained. Finally, we discuss our result and evaluate the process and timeline of the Hotel Identification 2022 challenge. All code used during the experiments related to this project can be found on GitHub[3].

2 METHOD

2.1 Data

The dataset provided by the Hotel-ID challenge consists of three directories:

- `train_images`: This directory contains 3.116 subdirectories, each representing one hotel. These subdirectories contain a combined 44.703 `jpg` images of hotel rooms. A visual inspection shows that these images were likely taken by visitors of the hotels rather than the hotel organisations themselves. Consequently, the images are frequently tilted, blurry, low quality, or incorrectly rotated (90 and even 180 degrees).
- `train_masks`: This directory consists of 4950 `png` images of arbitrary sizes, each containing one arbitrary rectangular red occlusion. The width and height of the occlusions seem randomly chosen between some bounds. These occlusions are present in the images in the test set, and may be combined with a training image to produce an image that is representative of a testing sample.

[1] https://www.kaggle.com/competitions/hotel-id-to-combat-human-trafficking-2022-fgvc9
[2] https://www.kaggle.com/competitions/hotel-id-2021-fgvc8
[3] https://github.com/tomaarsen/Hotel-ID-2022

MLiP'21-'22, Radboud University, Nijmegen, Netherlands

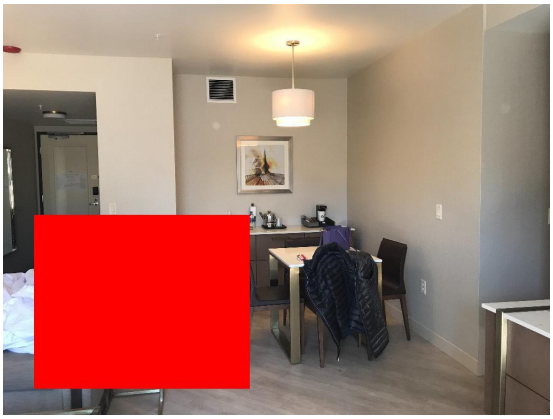

Figure 1: The only test sample provided by the Hotel-ID challenge dataset.

- `test_images`: This directory contains just one `jpg` image, which can be seen in Figure 1. On Kaggle, this directory will be filled with 5.000 similar test images. The images in this directory should be similar in quality to the training images, as they are likely drawn from the same source. However, in contrast to the training images, the test images have red occlusions arbitrarily added.
  Whenever a model is submitted to Kaggle, the reported score is computed with approximately 35% of the test data, while the final leaderboard ranking is determined by the remaining 65% of the test data.

For training, we partition the training images into two splits: training and validation, representing 90% and 10% of the original training images, respectively. The training partition is used to train the model, while the validation partition is only used for evaluation during training. When a model is trained and satisfactory, it may be uploaded to Kaggle, where it is then evaluated against the test data. During evaluation, the produced test embeddings will be compared against base embeddings of both the training and validation partitions.

2.2 Augmentations

2.2.1 Training augmentations. We apply augmentations in an online manner, meaning that augmentations are done during the training/testing process. The training and test/validation sets all have their own augmentation pipeline. More augmentations are applied on the training data to prevent overfitting and to try to close the performance gap between the validation/test sets and the train set. The augmentations that are applied to the training images are listed in order in Table 1. All augmentations are done using the Albumentations Python module [2].

	Augment	Prob	Note
1.	`RandomResizedCrop`	100%	
2.	`HorizontalFlip`	50%	
3.	`OneOf`: – `RandomBrightness` – `RandomContrast` – `RandomGamma`	50%	
4.	`ShiftScaleRotate`	30%	
5.	`CoarseDropout`	50%	Regular dropout
6.	`CoarseDropout`	100%	Simulates masks
7.	`Normalize`	100%	

Table 1: Augmentations used on the training images with their probabilities of being applied. The augmentations are listed in order of which they are applied. Prob stands for the probabilitity of application of that augmentation on a sample.

Note that we apply coarse dropout twice. The first time it is applied as a regular coarse dropout, coloring small squares in the given image with red. The second coarse dropout is always applied and is used to simulate the masks on the test samples. This augmentation adds a larger red rectangle to the given image. The minimum width and height of this rectangle is given by the resized image size divided by four. The maximum width and height of this rectangle is given by the resized image size divided by two. These sizes are based on the sizes of the given masks, and that of the mask in the given test image.

2.2.2 Validation and test augmentations. To get a representative idea of how well the model performs on the test set, we apply the coarse dropout simulating the masks on the validation images. This is not done for the test images, because these already contain a red mask. The following augmentations are used on the validation set: `Resize`, `CoarseDropout`, and `Normalize`. For the test set, only `Resize` and `Normalize` are applied.

Besides using a single resize for test images, we also experimented with cropping test images into multiple smaller images. Since the results of using this augmentation were not promising, we did not further experiment in this direction. A more detailed description as well as results can be found in Appendix D.

2.3 Models

2.3.1 Architecture. We have developed two separate PyTorch model architectures for the purposes of this challenge. These architectures center around ArcFace and CosFace, respectively, the former used as a prediction layer, and the latter as a loss function. Both of these model architectures contain an identical embedding layer. See Figures 2 and 3 for a general overview of the structure of our models during training.

In these figures, the batch serves as the input into the embedding layer, which contains a backbone and a section to regularise and reshape the embedding produced by the backbone into our desired embedding size. This final embedding is then passed through the prediction layer for ArcFace, or directly into the loss for CosFace. This loss then kickstarts backpropagation as denoted by the dashed arrow.

The backbone shown in these figures can be swapped out and replaced by any pre-trained PyTorch image classification model. See Section 2.6 for more information on which backbones were used with which model architectures.
Firstly, we will describe the core of these model architectures.

2.3.2 ArcFace. In the majority of our models, we use ArcFace [3] to compute the predictions. ArcFace is a classification loss function that aims to enforce larger distances between feature embeddings belonging to different classes, while at the same time minimizing the distances between feature embeddings belonging to the same class. It was originally designed for large scale face recognition tasks, which in general have large intra-class variations like age-gaps and pose variations. However, the hotel identification classification challenge from last year proved that ArcFace also works well on other classification tasks with large intra-class variations. The fact that most top ranking solutions of this challenge used ArcFace motivated us to use it in our solutions as well.

2.3.3 CosFace. Similar to ArcFace, CosFace [7] is classification loss that aims to enforce class separation of predictions. It was introduced prior to ArcFace, and differs primarily in that it adds a different kind of margin penalty: additive cosine margin, rather than additive angular margin as used in ArcFace. Further differences in our use case include that ArcFace is used as a prediction layer in the ArcFace architecture, while CosFace is used as a loss function in the CosFace architecture.

2.4 Evaluation metrics

During training, we rely on validation loss and prediction accuracy to evaluate model performance. The latter refers to the accuracy of correctly classifying the image to the hotel. Note that this can be a misleading metric, as the eventual hotel classification against the test set is done by comparing embeddings of training images against testing images. This means that the prediction layer of our network, which is included in the accuracy metric, does not impact the model performance in practice.

For the purposes of the challenge, the model is evaluated using Mean Average Precision at 5 (MAP@5) instead. A ranking of the top 5 hotels must be provided by the model, from which this metric is computed. The follow formula for MAP@5 was provided by the competition:

$$MAP@5 = \frac{1}{U}\sum_{u=1}^{U}\sum_{k=1}^{min(n,5)} P(k) \times rel(k)$$

where U is the number of images, $P(k)$ is the precision at cutoff k, n is the number of predictions for that image, and $rel(k)$ denotes relevance with 1 if the hotel at rank k is correct, and 0 otherwise.

A hotel is only considered relevant once, so duplicate occurrences of a hotel in the ranking do not contribute to a higher score.

2.5 Training strategy

This section describes the training strategy used to obtain our results. A more general overview of the corresponding pipeline for the two model architectures can be seen in Figures 2 and 3. Using these two model architectures, we train 7 different models using different baselines and hyperparameters. All models are trained

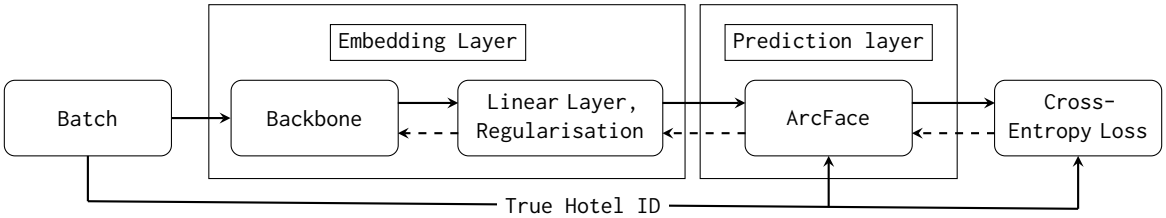

Figure 2: High level overview of the general model architecture using ArcFace. The dashed arrow denotes backpropagation.

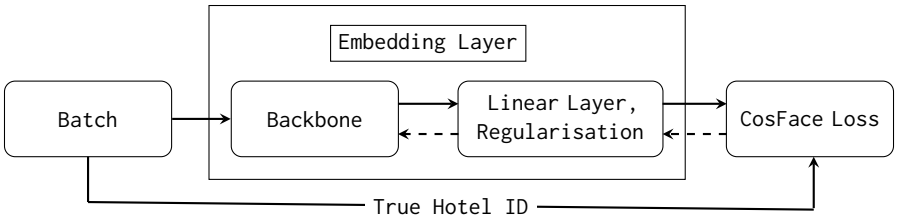

Figure 3: High level overview of the general model architecture using CosFace. The dashed arrow denotes backpropagation.

using stochastic gradient descent with a cosine annealing scheduler to update the learning rate over time. The exact parameters that were used can be found in Table 4 of Appendix E.

2.5.1 Ranking strategy. To obtain a ranking of hotels for a given test image and model, we first produce base embeddings by passing both the training and validation images through only the embedding layer of the model. We compare the embedding of the test image with the base embeddings using cosine similarity. The unique hotel IDs of base embeddings that are most similar to the test image embedding are then used to generate the final prediction.

It is important to note that in order to compute the embeddings, all images are augmented using the test augmentation pipeline (i.e. resize followed by normalization).

2.5.2 Ensemble. Several models have been trained to be included in our experiments and submissions. The exact details of these models, including the backbone names, hyperparameters and training statistics can be found in Table 5 in Appendix E. See Section 2.6 for an elaboration of how we came to train exactly these models. The pre-trained backbones for these models are downloaded from timm [8] using the corresponding backbone names. No layers of these backbones are frozen in our experiments, i.e. all layers can be fine-tuned, as we found freezing layers to be quite detrimental to our experimental results.

Different combinations of these models have been ensembled in order to create robust and effective models, while learning about the performance consequences of including or excluding certain models. The ensembling is performed by summing the cosine similarities for all models together using equal weights, producing a vector with a score per image indicating whether the models agree that the image produced a similar embedding as the test embedding. We opt to use equal weights rather than relying on the public leaderboard results to determine custom weights, as we did not want to overfit on the public leaderboard. We feel this goes against the spirit of the competition, and could hurt generalisability in practice.

2.5.3 Hardware. The training was performed on a cluster running Ubuntu. On it, PyTorch was given access to 2 Nvidia RTX 2080 Ti GPUs with 11GB memory each, 6 Xeon CPUs, and as much memory as was required. The base embeddings described in Section 2.5.1 were also generated on this cluster.

Beyond that, the evaluation was performed on Kaggle using the GPU Accelerator. The smaller ensembles of 4 or less models were able to perform all computations on the GPU, causing the evaluation time to only be an hour or less. For the more expensive ensembles, some computations had to be moved to the CPU for CUDA memory allocation reasons, causing those evaluations to take upwards of 5 hours.

2.6 Training process

Now that the training and submission strategy has been explained, we can go over our training process throughout this challenge. For reference, all trained models used in our ensembles can be found in Table 5 in Appendix E. We first implemented the ArcFace model architecture from Figure 2. With this architecture, the first backbone we used was `eca_nfnet_l0` [1], inspired by the 8th place solution[4] of the 2021 Hotel-ID challenge by Kaggle user michaln. This resulted in a solid 44.8% validation accuracy after 7 hours of training. We then moved to the larger `eca_nfnet_l2` backbone [1], which improved to 59.2% validation accuracy after 18 or so hours. This was our first model m_1.

Based on the success of the first model, we decided to use ArcFace in combination with other backbones which performed well on last-year's Hotel-ID challenge. The second backbone we experimented with was the `efficientnet_b3` backbone [6]. Although this model did not perform as well as the first model, we were motivated to use it in an ensemble, which ended up paying off.

For the third model we used the `eca_nfnet_l2` backbone again due to its solid performance. However, in order to obtain predictions

[4]https://www.kaggle.com/competitions/hotel-id-2021-fgvc8/discussion/242207

partially independent of that of our first model, we used CosFace to compute the loss instead of ArcFace. After training this model, we started creating ensembles and seeing good results from this on the Kaggle leaderboard. The results we saw motivated us to use more models in a single ensemble, and experiment with completely different network structures, in the hope that these models produce different errors than our existing models. This would be beneficial to our ensembles. As such, the fourth model we introduced in our ensemble was a vision transformer called `vit_small_patch16_-138` [4]. After running some experiments with this m_4, we noticed that it had not fully converged in training yet, so we retrained it for longer to create m_5.

Since the performance of the ensembles kept increasing as we introduced more models, we decided to also try out `regnety_-120` [5] and `resnet101e` [9], two models used by the first place solution[5] of the 2021 Hotel-ID challenge by Kaggle user Kohei.

As we experimented more with ensembles of different types, we started noticing diminishing returns on adding new models. We also encountered issues with CUDA memory on Kaggle when we used too many models. This marked a natural stopping point.

Throughout the training process, we also experimented with different augmentations like `CLAHE`, `InvertImg`, `Perspective`, `RandomCrop`, `OpticalDistortion` and `GridDistortion`. We also experimented with changing the probabilities of the augmentations from Table 1. None of these showed a noticeable increase or decrease in performance.

We also performed experiments with different learning rates, optimizers and schedulers, but found all attempts to be worse than what we had started with, i.e. the hyperparameters from Table 4. These values were inspired by our previous paper on the TinyVoxCeleb Speaker Recognition challenge.

3 RESULTS & DISCUSSION

3.1 Ensemble performance

We experimented with ensembles of up to six different models, where the models are selected from a set of seven models ($\{m_1, \ldots, m_7\}$). For an overview of the models, their corresponding backbones, hyperparameters, and training statistics, we refer to Table 5 of the Appendix. The MAP@5 scores for all of our considered ensembles are listed in Table 2. For each number of considered models n, the best MAP@5 score is denoted by bold text.

In general, we can see that the performance increases by considering more models in the ensemble, where the largest increase in performance is obtained by considering three instead of one model. The best performance is achieved by our ensemble consisting of five models, which achieved a MAP@5 of 0.576. We also see that the performance on the public test set seems to be indicative of the performance on the private test set.

Note that although the MAP@5 score increases as the number of models in the ensemble is incremented, we decided not to use more than 6 models in a single ensemble. The main reason for this is that the training time and required memory grows approximately linear with every model we add, while the MAP@5 score seems to converge at five or six models in a single ensemble. We argue that adding more models would result in a very limited increase in score, if not a decrease in score, and is therefore not worthwhile.

We submitted the ensembles $\{m_1, m_2, m_3, m_5\}$ with a public MAP@5 of 0.574 and $\{m_1, m_2, m_3, m_5, m_6\}$ with a public MAP@5 of 0.576. In the end, this resulted in a final private leaderboard MAP@5 of 0.568, which yielded us a 7th place entry.

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	Publ.	Priv.	
$n=1$	×							**0.520**	**0.511**	
		×						0.504	0.504	
			×					0.499	0.500	
				×				0.480	0.488	
					×			0.470	0.472	
						×		0.436	0.443	
							×	0.421	0.425	
$n=3$		×	×	×				**0.568**	0.556	
	×		×	×				0.562	0.555	
	×	×		×				0.564	**0.561**	
	×	×	×					0.563	0.550	
		×	×		×			0.566	0.555	
$n=4$	×	×	×	×				0.573	**0.563**	
	×	×	×		×			**0.574**	0.561	←
	×	×	×			×		0.569	0.562	
	×	×	×				×	0.564	0.560	
		×	×		×	×		0.568	0.561	
$n=5$	×	×	×		×	×		**0.576**	**0.568**	←
$n=6$	×	×	×		×	×	×	**0.571**	**0.569**	

Table 2: Ensemble performance, segmented by number of models in the ensemble. Publ. denotes the public leaderboard MAP@5 scores, while Priv. denotes the private ones. Higher is better. Bold refers to the best performance per segment, while underscore shows the second best performance per segment. The two rows with arrows were submitted to count towards our final leaderboard score. m_1 is `eca_nfnet_l2` with ArcFace, m_2 is `efficientnet_b3` with ArcFace, m_3 is `eca_nfnet_l2` with CosFace, m_4 is `vit_small_patch16_384` (20 epochs), m_5 is `vit_small_-patch16_384` (50 epochs) with ArcFace, m_6 is `regnety_120` (60 epochs) with ArcFace, m_7 is `resnest101e` (60 epochs) with ArcFace.

3.2 Evaluation augments

Our hypothesis was that the training and test images should be made as similar as possible during ranking, and thus the validation augments should be run on the training images so that both sets of images have red obfuscations. However, as mentioned in Section 2.5.1, we end up using the test augmentation pipeline to preprocess all of our images before producing embeddings. This is because our experiments show that this configuration produced the best performance, as can be seen in Table 3.
We theorize that this unexpected effect is caused by 2 factors:

(1) An obfuscated training image is unable to embed potentially useful hotel-identifying information that is hidden behind the obfuscation.

[5] https://www.kaggle.com/competitions/hotel-id-2021-fgvc8/discussion/242087

Training images	Test images	MAP@5
Validation augments	Validation augments	0.480
Validation augments	Test augments	0.490
Test augments	Test augments	**0.496**

Table 3: Consequences of using different augmentation techniques during final prediction. These tests are performed using a weaker variant of the m_1 model, using the `eca_nfnet_l0` backbone, and the scores are on the public leaderboard.

(2) The trained models are learned to ignore the red obfuscation in the embedding.

As a result of these two factors, a training image with as much information readily available to be placed in the embeddings should produce better results, and that is what can be seen in Table 3.

4 CONCLUSION

Using our described ensemble model and training pipeline, we were able to obtain a 0.568 MAP@5 score on the private leaderboard, resulting in the 7th-entry on the Hotel-ID to Combat Human Trafficking 2022 Kaggle challenge. The most important factors that allowed for this score are proper backbone models and class separating loss functions like ArcFace and CosFace. Furthermore, the data augmentation pipeline used on the training set helped to improve the score significantly.

A AUTHOR CONTRIBUTIONS

Aarsen was responsible for setting up the codebase given the initial skeleton code, and implementing the ArcFace and CosFace architectures given existing implementations. Aarsen also worked on ensembling and setting up the pipeline of training and evaluating models.

Berns mainly considered experimenting with cropping test images into multiple smaller sub-images removing the mostly red squares. Besides this, Berns experimented with different hyperparameters for the backbones, schedulers, and optimizers.

B EVALUATION OF THE PROCESS

Since we had a good experience with the CN clusters from the previous challenge and we both prefer to work in Python files instead of notebooks, we decided to perform the training of models on the CN clusters. As a base for our code, we used the skeleton code provided at the start of the speaker recognition challenge. This skeleton code was familiar to us, and allowed us to quickly start implementing potential solutions.

The major difficulty with training the models this way was that the trained model, the base embeddings, the code we use for evaluating a model, the pre-trained `timm` models, as well as pip dependencies all had to be uploaded to Kaggle. In order to do this, we used private Kaggle datasets, in which we uploaded the code, trained models and wheel files. This implied that each time we wanted to get a MAP@5 score on the test set, we had to download all the necessary files from the CN clusters and upload them to Kaggle. This is a relatively slow procedure, especially compared to the ease of getting a test score in the previous challenge. Other than this major difficulty, we never ran into any other problems while working with Kaggle.

C EVALUATION OF THE SUPERVISION

There was a helpful teacher meeting towards the start of the challenge that was able to point us in the right direction. Beyond that, there were regular appointments with a coach. A second group would participate in the coach meetings, which was particularly interesting as it was inspiring and helpful to discuss with another team. Both type of meetings felt very accessible and relaxed.

D ALTERNATIVE AUGMENTATIONS

During the project, we experimented with multiple augmentations pipelines. Most different from our best performing augmentation pipeline (see Table 1), is the case in which test images are cropped into multiple smaller sub-images, where images consisting of mostly (more than 50%) red pixels are discarded.

During prediction, we used the multiple smaller images to obtain multiple predictions. The final prediction for a given image is then obtained by taking the average over these predictions. This is done by adding the cosine similarity between train embeddings and a crop for all crops belonging to the same picture. Using `eca_nfnet_l2` as a backbone and five non-overlapping crops for every test image, we obtained a MAP@5 of 0.248 on the public leaderboard. The same backbone in combination with eight partially overlapping crops resulted in a MAP@5 of 0.489.

Although we expected a performance increase by not considering mostly red images, the obtained results were inferior compared to the results obtained by basing the prediction on a single resized image using the same model. We hypothesize that this inferior performance can be explained by the fact that a cropped image contains less information than a resized image. For example, we noted that often a cropped image would only consist of a wall, floor or ceiling. Predictions based on such images are most likely far from desired, and therefore worsen the average prediction based on all crops.

E TRAINED MODELS & HYPERPARAMETERS

Embedding size	4096
Optimizer	`SGD`
– Learning rate	0.001
– Momentum	0.9
– Weight decay	0.0005
Scheduler	`CosineAnnealing`
– Minimum learning rate	0.0

Table 4: Hyperparameters shared between all models. The remainder of the hyperparameters for each trained model can be found in Table 5.

m_1	Backbone name	`eca_nfnet_l2` [1]
	Architecture	`ArcFace` architecture
	Backbone param count	56.720.000
	Image size	512×512
	Batch size	4
	Epoch count	20
	Training time	18:15:00
	Cross-Entropy Val. loss	3.070
	Val. accuracy	59.2%

m_2	Backbone name	`efficientnet_b3` [6]
	Architecture	`ArcFace` architecture
	Backbone param count	12.230.000
	Image size	512×512
	Batch size	8
	Epoch count	20
	Training time	10:30:00
	Cross-Entropy Val. loss	4.060
	Val. accuracy	46.1%

m_3	Backbone name	`eca_nfnet_l2` [1]
	Architecture	`CosFace` architecture
	Backbone param count	56.720.000
	Image size	512×512
	Batch size	4
	Epoch count	20
	Training time	20:00:00
	CosFace Val. loss	13.10
	Val. accuracy	N/A

m_4	Backbone name	`vit_small_patch16_384` [4]
	Architecture	`ArcFace` architecture
	Backbone param count	22.200.000
	Image size	384×384
	Batch size	16
	Epoch count	20
	Training time	8:00:00
	Cross-Entropy Val. loss	4.480
	Val. accuracy	42.2%

m_5	Backbone name	`vit_small_patch16_384` [4]
	Architecture	`ArcFace` architecture
	Backbone param count	22.200.000
	Image size	384×384
	Batch size	16
	Epoch count	50
	Training time	14:45:00
	Cross-Entropy Val. loss	3.260
	Val. accuracy	55.7%

m_6	Backbone name	`regnety_120` [5]
	Architecture	`ArcFace` architecture
	Backbone param count	51.820.000
	Image size	224×224
	Batch size	8
	Epoch count	60
	Training time	18:00:00
	Cross-Entropy Val. loss	3.740
	Val. accuracy	53.5%

m_7	Backbone name	`resnest101e` [9]
	Architecture	`ArcFace` architecture
	Backbone param count	48.280.000
	Image size	256×256
	Batch size	8
	Epoch count	60
	Training time	20:30:00
	Cross-Entropy Val. loss	3.820
	Val. accuracy	53.1%

Table 5: Our trained models with the corresponding backbones, hyperparameters and training statistics. See Table 4 for the hyperparameters that are shared between all of these models.

REFERENCES

[1] Andy Brock, Soham De, Samuel L Smith, and Karen Simonyan. 2021. High-performance large-scale image recognition without normalization. In *International Conference on Machine Learning*. PMLR, 1059–1071.

[2] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. 2020. Albumentations: Fast and Flexible Image Augmentations. *Information* 11, 2 (2020).

[3] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. 2019. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4685–4694.

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929 [cs.CV]

[5] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollar. 2020. Designing Network Design Spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[6] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 6105–6114.

[7] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. 2018. CosFace: Large Margin Cosine Loss for Deep Face Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[8] Ross Wightman. 2019. PyTorch Image Models. https://github.com/rwightman/pytorch-image-models. https://doi.org/10.5281/zenodo.4414861

[9] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Haibin Lin, Zhi Zhang, Yue Sun, Tong He, Jonas Mueller, R. Manmatha, Mu Li, and Alexander Smola. 2020. ResNeSt: Split-Attention Networks. arXiv:2004.08955 [cs.CV]