Assignment

AnimalCLEF25 @ CVPR-FGVC & LifeCLEF

Vișan Ionuț

1. Introduction

The AnimalCLEF 2025 challenge [1], hosted on Kaggle, is a leading competition in the domain of computer vision and biodiversity informatics. It focuses on the fine-grained task of individual animal identification across multiple species, under real-world, unconstrained conditions. Participants are required to develop robust machine learning models capable of accurately recognizing individual animals from photographic data, despite variations in image quality, angle, lighting, and environmental context.

The competition provides a large-scale, annotated dataset to support the training and evaluation of algorithms, encouraging the use of advanced techniques in deep learning, metric learning, and hierarchical classification.

2. Data Analysis

The main dataset is split into two main subsets:

- Database – Contains labeled data about the individuals.
- Query – Consists of images for identification, where the task is to determine whether the individual is known (present in the database) or new (not present in the database).

The dataset is organized into structured directories to allow easy access and processing. It consists of labeled images for known individuals (*Database*) and unlabeled images (*Query*) requiring identification, metadata, and a sample submission file.

This dataset is all about individual animal identification of the following three species: loggerhead sea turtles (Zakynthos, Greece), salamanders (Czech Republic), Eurasian lynxes (Czech Republic).

The dataset contains 15209 images (database – 13074, query – 2135) and a metadata.csv file which contains multiple columns (image_id, identity, path, date, orientation, species, split, dataset).

2.1. Data Split

We will use the "database" dataset to train the model, as the "query" dataset does not contain labels and is intended for the evaluation of the results.

In this code, the dataset is split as follows:

1. **Identify rare identities:** All images whose identity appears only once (single-occurrence identities) are isolated into a separate subset (317).
2. **Separate normal identities:** Images belonging to identities that appear at least twice are separated for stratified sampling.
3. **Stratified split:** The normal identities are split into 80% training and 20% testing sets using train_test_split, with stratification based on the “identity” column to preserve the class distribution. A random seed (random_state=42) is set for reproducibility.
4. **Training set:** The training set consists of 80% of the normal identities plus all single-occurrence images.
5. **Testing set:** The testing set contains only the remaining 20% of the normal identities; single-occurrence identities are **not** included in the testing set.

Now we have (number of images):

Train set: 10522

Test set: 2552

Split ratio: 0.2425

2.1. Dataloaders

Custom AnimalDataset objects are created from CSV metadata, applying image transformations and encoding identities, then wrapped into PyTorch DataLoaders with batching, shuffling (for training), and a custom collate_fn to handle image-label-path-identity grouping.

3. Related Work

As the competition is still in progress, there are currently few available notebooks or discussions showcasing strong-performing implementations. Most existing approaches, such as clustering methods or simple convolutional networks, have so far yielded suboptimal results.

I decided to directly explore an approach known for achieving remarkable results: visual transformers.

Vision Transformers (ViTs) [2] are a type of deep learning model that applies the transformer architecture, originally designed for natural language processing, to image data. Instead of using convolutional layers like traditional CNNs, ViTs split an image into fixed-size patches, flatten them, and process them as a sequence of tokens. This allows the model to capture long-range dependencies and global context more effectively, leading to strong performance on various computer vision tasks, especially when trained on large datasets.

We will also use architectures such as ConvNeXt [3] and Swin Transformer [4] for building our models, alongside classical Vision Transformers. ConvNeXt is a modernized convolutional network that incorporates design elements from transformers while maintaining a purely convolutional structure. Swin Transformer, on the other hand, introduces a hierarchical architecture with local self-attention windows, making it more efficient and scalable compared to classical ViTs, which apply global self-attention across all image patches without modeling locality.

4. Implementation

The approach involves fine-tuning a pretrained ViT/ConvNeXt/Swin model on the AnimalCLEF dataset. The images are heavily augmented during training to improve generalization, using a combination of random cropping, flipping, rotation, color jittering, and advanced augmentation policies. The model is optimized using AdamW with a linear learning rate scheduler and a warmup strategy. We monitor both validation loss and accuracy at each epoch, saving the model checkpoint whenever both metrics improve. The final setup focuses on adapting a strong pretrained backbone to the fine-grained animal identification task.

I will now present several training configurations and their corresponding results.

4.1. Configuration 1

resize dimension = 384x384

augmentation = no

batch size = 32

model = “vit_base_patch16_384”

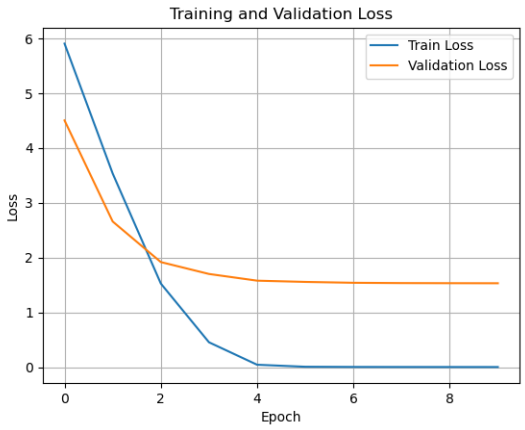
epochs = 10

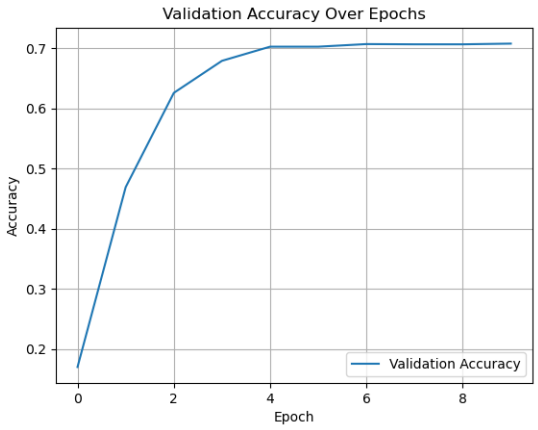
learning rate = 5e-5

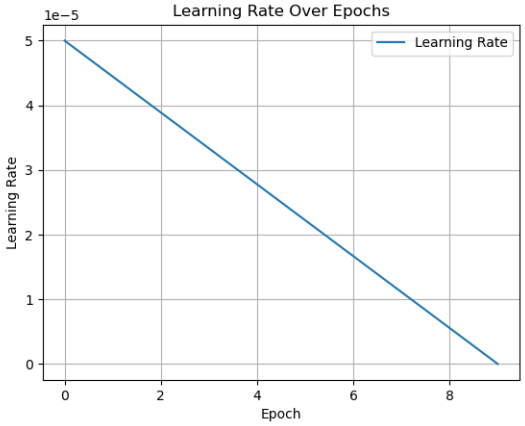
weight decay = 0.01

scheduler = linear (with warmup – 0.1)

Results:







4.2. Configuration 2

resize dimension = 384x384

augmentation = no

batch size = 16

model = “vit_large_patch16_384”

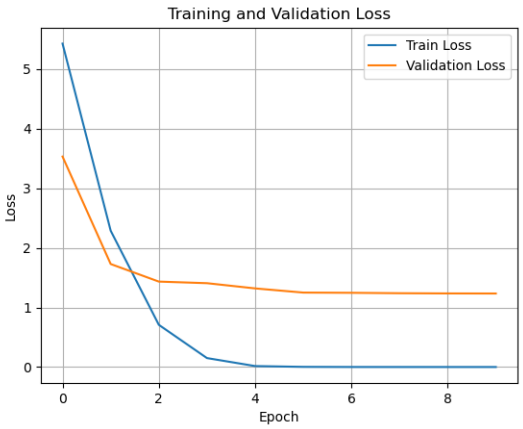
epochs = 10

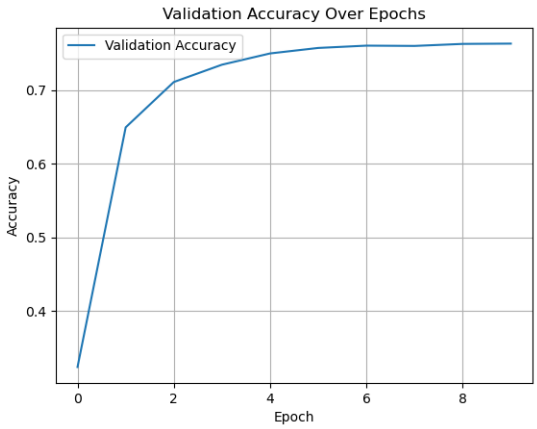
learning rate = 5e-5

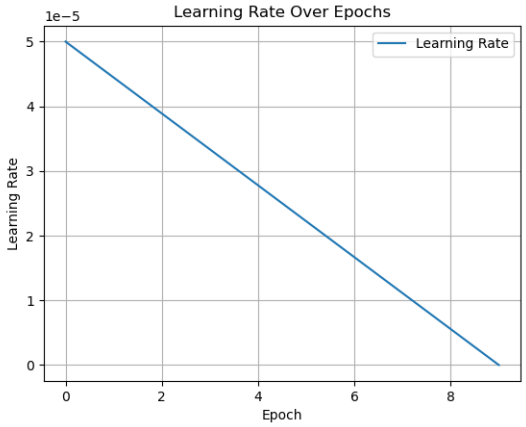
weight decay = 0.01

scheduler = linear (with warmup – 0.1)

Results:







epoch	train_loss	val_loss	val_acc	learning_rate
1	5.429086	3.532917	0.32406	5.00E-05
2	2.288315	1.72938	0.649295	4.44E-05
3	0.708581	1.434405	0.710815	3.89E-05
4	0.150383	1.407622	0.734326	3.33E-05
5	0.016526	1.320042	0.749608	2.78E-05
6	0.003086	1.249966	0.757053	2.22E-05
7	0.000996	1.246246	0.760188	1.67E-05
8	0.000732	1.239191	0.759796	1.11E-05
9	0.000619	1.235722	0.762539	5.56E-06
10	0.000559	1.234373	0.762931	0

4.3. Configuration 3

resize dimension = 384x384

augmentation = RandomResizedCrop, RandomHorizontalFlip, RandomRotation, ColorJitter, AutoAugment (ImageNet policy), RandomErasing

batch size = 16

model = “vit_large_patch16_384”

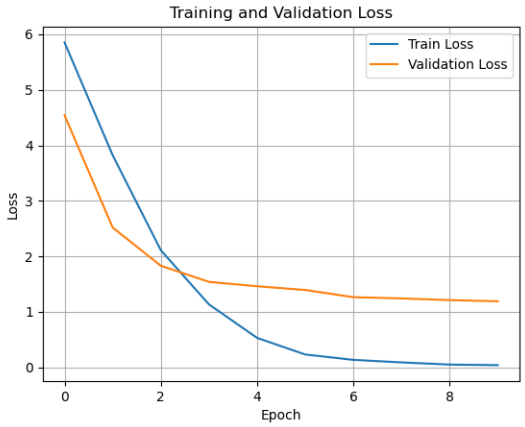
epochs = 10

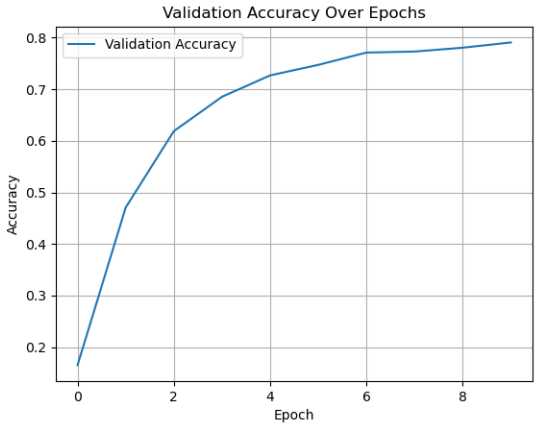
learning rate = 8e-5

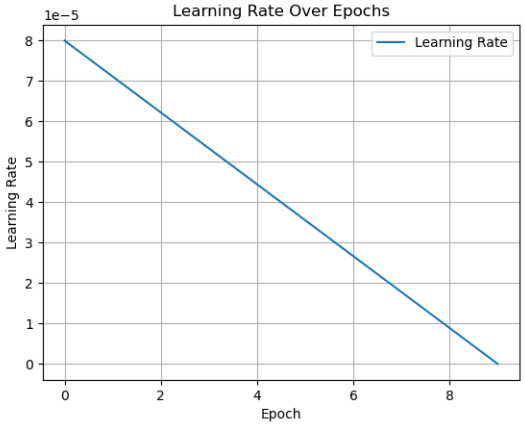
weight decay = 0.01

scheduler = linear (with warmup – 0.1)

Results:







epoch	train_loss	val_loss	val_acc	learning_rate
1	5.854455	4.542987	0.164969	8.00E-05
2	3.821325	2.518633	0.470611	7.11E-05
3	2.109785	1.829655	0.61873	6.22E-05
4	1.133664	1.539898	0.685345	5.33E-05
5	0.530054	1.460773	0.726881	4.44E-05
6	0.231077	1.393552	0.747257	3.56E-05
7	0.134431	1.264359	0.77116	2.67E-05
8	0.088636	1.241163	0.773119	1.78E-05
9	0.049385	1.211593	0.780564	8.89E-06
10	0.039044	1.189801	0.790752	0

4.4. Configuration 4

resize dimension = 384x384

augmentation = RandomResizedCrop, RandomHorizontalFlip, RandomRotation, ColorJitter, AutoAugment (ImageNet policy), RandomErasing

batch size = 16

model = “vit_large_patch16_384”

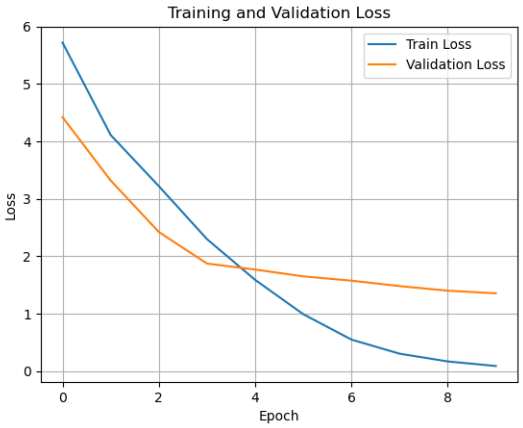
epochs = 10

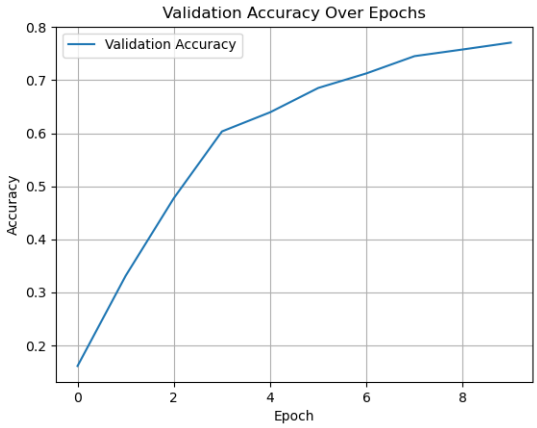
learning rate = 2e-4

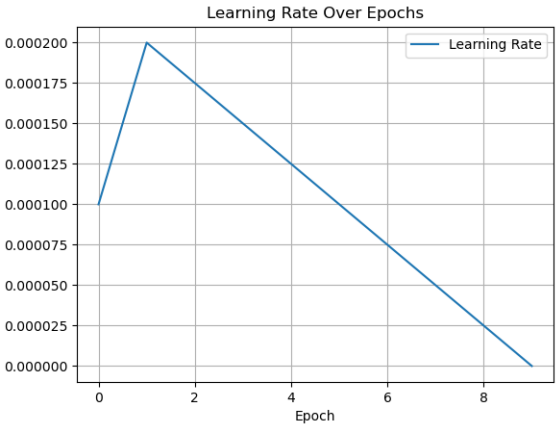
weight decay = 0.01

scheduler = linear (with warmup – 0.2)

Results:







epoch	train_loss	val_loss	val_acc	learning_rate
1	5.720766	4.422295	0.161442	0.0001
2	4.112145	3.31994	0.331505	0.0002
3	3.221721	2.423418	0.477665	0.000175
4	2.299755	1.873858	0.603448	0.00015
5	1.589857	1.769489	0.639498	0.000125
6	0.992501	1.651043	0.685345	0.0001
7	0.550343	1.576184	0.712774	7.50E-05
8	0.304945	1.481638	0.745298	5.00E-05
9	0.170039	1.401846	0.757837	2.50E-05
10	0.090172	1.355726	0.770768	0

4.5. Configuration 5

resize dimension = 384x384

augmentation = no

batch size = 16

model = “vit_huge_patch16_384”

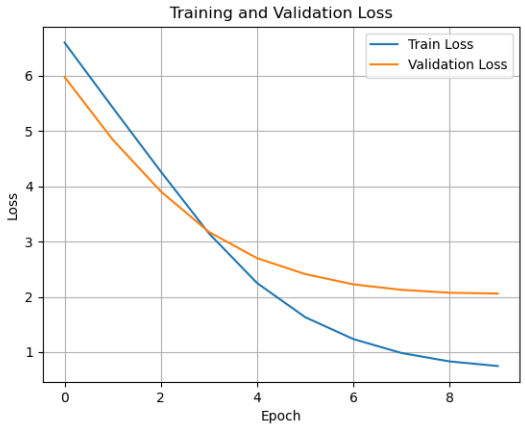
epochs = 10

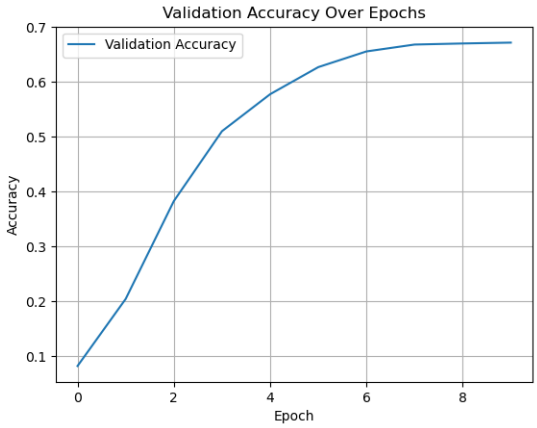
learning rate = 5e-5

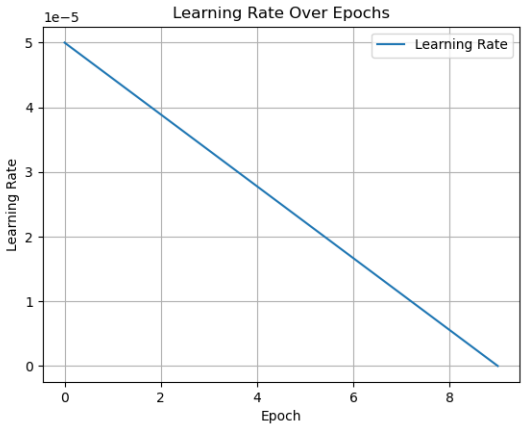
weight decay = 0.01

scheduler = linear (with warmup – 0.1)

Results:







epoch	train_loss	val_loss	val_acc	learning_rate
1	6.599495	5.977871	0.081897	5.00E-05
2	5.424447	4.845208	0.204154	4.44E-05
3	4.266296	3.907051	0.382445	3.89E-05
4	3.147225	3.173598	0.509796	3.33E-05
5	2.248571	2.698761	0.577194	2.78E-05
6	1.631079	2.411274	0.626567	2.22E-05
7	1.235039	2.225065	0.655172	1.67E-05
8	0.98322	2.126686	0.667712	1.11E-05
9	0.830694	2.073604	0.669671	5.56E-06
10	0.748063	2.059534	0.671238	0

4.6. Configuration 6

resize dimension = 384x384

augmentation = RandomResizedCrop, RandomHorizontalFlip, RandomRotation, ColorJitter, AutoAugment (ImageNet policy), RandomErasing

batch size = 8

model = “convnext_xlarge.fb_in22k_ft_in1k_384”

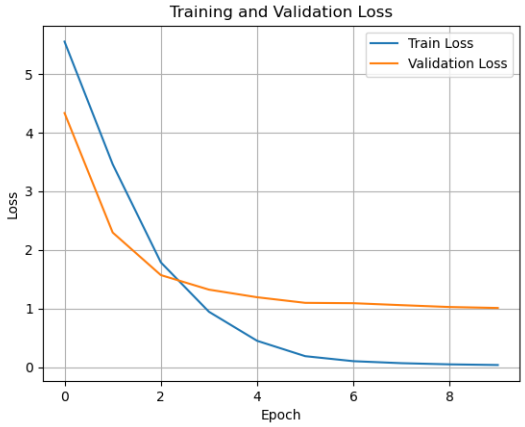
epochs = 10

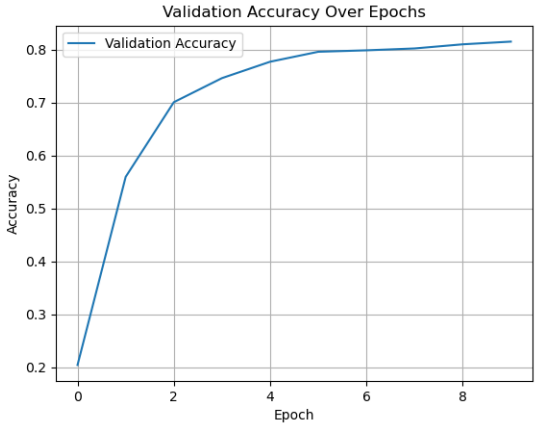
learning rate = 8e-5

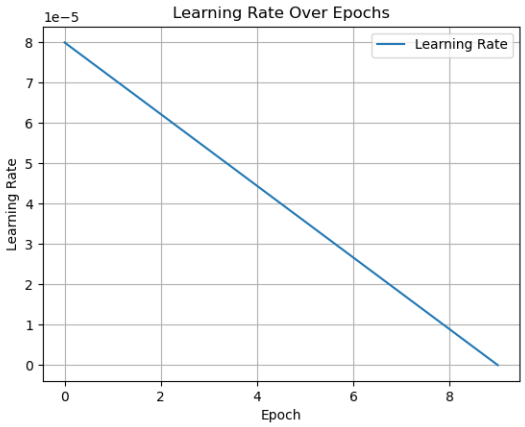
weight decay = 0.01

scheduler = linear (with warmup – 0.1)

Results:







epoch	train_loss	val_loss	val_acc	learning_rate
1	5.554994	4.33619	0.204545	8.00E-05
2	3.462827	2.298031	0.559953	7.11E-05
3	1.789216	1.571954	0.701019	6.22E-05
4	0.945808	1.323647	0.746473	5.33E-05
5	0.450624	1.193603	0.777429	4.44E-05
6	0.188344	1.098227	0.796238	3.56E-05
7	0.103085	1.092235	0.798981	2.67E-05
8	0.069024	1.058875	0.802508	1.78E-05
9	0.048916	1.027089	0.810345	8.89E-06
10	0.037705	1.010882	0.815439	0

4.7. Configuration 7

resize dimension = 384x384

augmentation = RandomResizedCrop, RandomHorizontalFlip, RandomRotation, ColorJitter, AutoAugment (ImageNet policy), RandomErasing

batch size = 8

model = “convnext_xlarge.fb_in22k_ft_in1k_384”

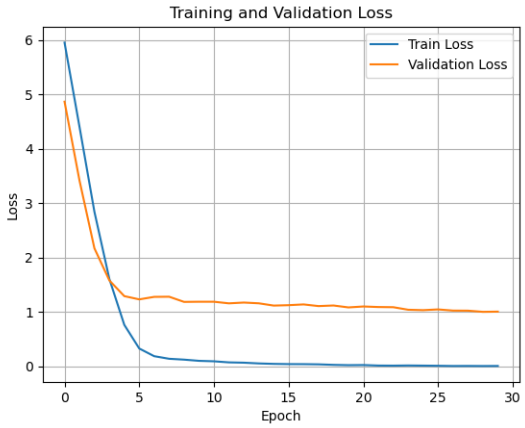
epochs = 30

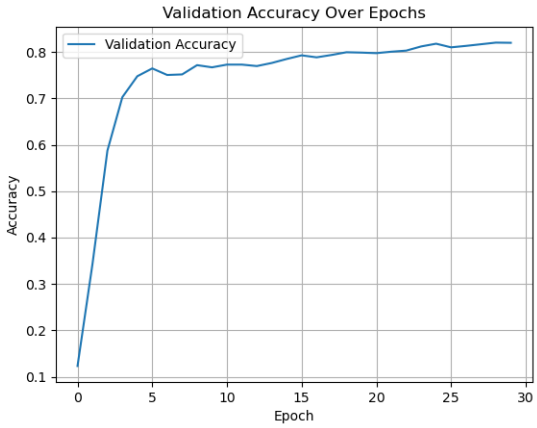
learning rate = 8e-5

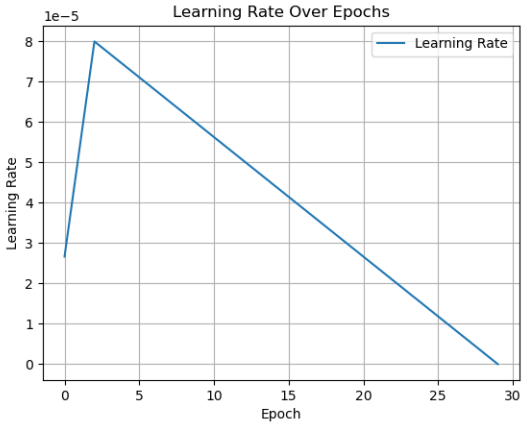
weight decay = 0.01

scheduler = linear (with warmup – 0.1)

Results:







epoch	train_loss	val_loss	val_acc	learning_rate
1	5.95136	4.86577	0.12304	2.67E-05
2	4.4059	3.41841	0.34287	5.33E-05
3	2.84234	2.17129	0.58738	8.00E-05
4	1.60781	1.58053	0.70259	7.70E-05
5	0.76431	1.29236	0.74765	7.41E-05
6	0.33008	1.2327	0.7645	7.11E-05
7	0.18872	1.27889	0.75039	6.81E-05
8	0.1411	1.28146	0.75157	6.52E-05
9	0.12526	1.18651	0.77155	6.22E-05
10	0.1032	1.1894	0.76685	5.93E-05
11	0.09476	1.1894	0.77273	5.63E-05
12	0.07486	1.15989	0.77273	5.33E-05
13	0.06917	1.17441	0.76959	5.04E-05
14	0.05554	1.16056	0.77625	4.74E-05
15	0.04735	1.11861	0.78487	4.44E-05
16	0.04322	1.12611	0.79271	4.15E-05
17	0.04227	1.13998	0.7884	3.85E-05
18	0.0388	1.10934	0.7935	3.56E-05
19	0.02936	1.12023	0.79937	3.26E-05
20	0.02379	1.08496	0.79859	2.96E-05
21	0.02599	1.10195	0.79741	2.67E-05
22	0.01649	1.09208	0.80055	2.37E-05
23	0.01501	1.08884	0.8029	2.07E-05
24	0.01802	1.04191	0.81191	1.78E-05
25	0.01538	1.0346	0.81779	1.48E-05
26	0.01264	1.04805	0.80995	1.19E-05
27	0.0087	1.02665	0.81309	8.89E-06
28	0.00975	1.02524	0.81661	5.93E-06
29	0.00813	1.00358	0.82014	2.96E-06
30	0.00937	1.00737	0.81975	0

4.8. Configuration 8

resize dimension = 384x384

augmentation = RandomResizedCrop, RandomHorizontalFlip, RandomRotation, ColorJitter, AutoAugment (ImageNet policy), RandomErasing

batch size = 8

model = “swin_large_patch4_window12_384”

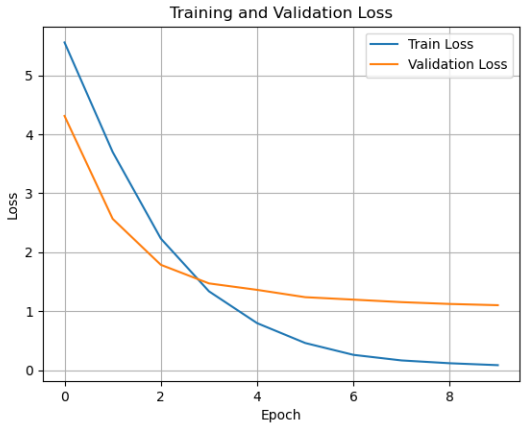
epochs = 10

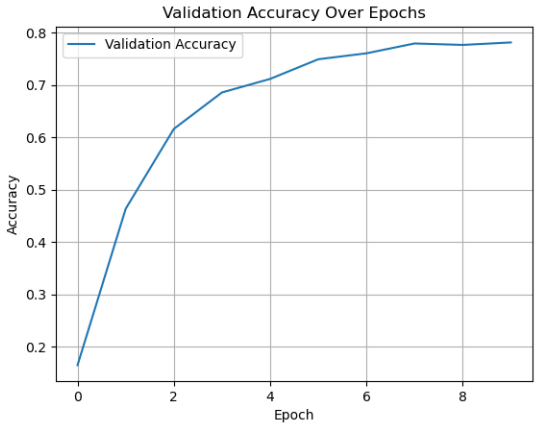
learning rate = 8e-5

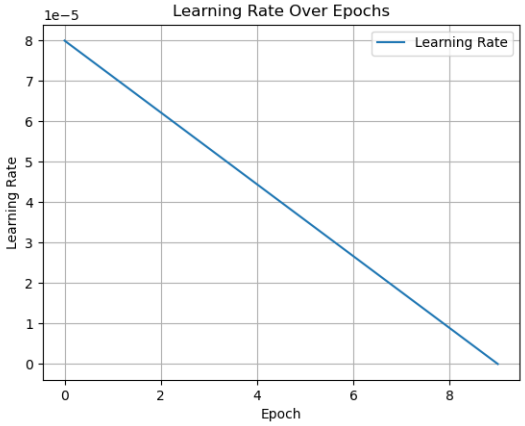
weight decay = 0.01

scheduler = linear (with warmup – 0.1)

Results:







epoch	train_loss	val_loss	val_acc	learning_rate
1	5.556254	4.310629	0.164577	8.00E-05
2	3.699614	2.567253	0.463558	7.11E-05
3	2.233528	1.78865	0.616379	6.22E-05
4	1.339183	1.473503	0.686129	5.33E-05
5	0.800015	1.364044	0.711991	4.44E-05
6	0.463155	1.239894	0.749608	3.56E-05
7	0.262838	1.198547	0.760972	2.67E-05
8	0.168196	1.156086	0.779781	1.78E-05
9	0.120594	1.125174	0.777038	8.89E-06
10	0.088192	1.104203	0.78174	0

5. Submissions

The evaluation uses two metrics: BAKS (balanced accuracy on known samples) and BAUS (balanced accuracy on unknown samples). The final score is the geometric mean of BAKS and BAUS to prevent biased results from trivial predictions.

To determine whether an image belongs to a known class or should be classified as a "new individual," we will use a threshold.

1. 0.34255 - Configuration 2 (4.2.) and threshold = 0.68

2. 0.36100 – Configuration 3 (4.3.) and threshold = 0.68

3. 0.35781 – Configuration 3 (4.3.) and threshold = 0.7

4. 0.36177 – Configuration 3 (4.3.) and threshold = 0.65

5. 0.36007 – Configuration 6 (4.6.) and threshold = 0.65

6. 0.36751 – Configuration 6 (4.6.) and threshold = 0.6

7. 0.43519 – Configuration 7 (4.7.) and threshold = 0.68

8. 0.43492 – Configuration 7 (4.7.) and threshold = 0.7

9. 0.44326 - Configuration 7 (4.7.) and threshold = 0.65

10. 0.43576 - Configuration 7 (4.7.) and threshold = 0.6

The competition has a total of 152 registered teams, with 16 teams scoring 0, resulting in 136 active and valid teams.

I managed to rank 37th out of 136 valid teams (27%), with a score of 0.44326.

6. Future approaches

In the upcoming experiments, I plan to explore additional models and configurations, implement ensemble strategies, and apply various dataset modifications, including alternative augmentations and the integration of new data.

7. References

[1] - https://www.kaggle.com/competitions/animal-clef-2025/overview

[2] – "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" by Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, et al. - https://arxiv.org/pdf/2010.11929

[3] – "A ConvNet for the 2020s" by Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, Saining Xie - https://arxiv.org/pdf/2201.03545

[4] – "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows" by Ze Liu, Yutong Lin, Yue Cao, Han Hu, et al. - https://arxiv.org/pdf/2103.14030