Research workshops (Warsztaty badawcze). Neural Networks.

TRANSFER LEARNING FOR CNN AND TRANSFORMER ARCHITECTURES. CURRICULUM LEARNING FOR CNN

* Authors in alphabetical order *
Katsiaryna Bokhan
Milanna Pahasian
Rafał Pyzowski
Jakub Sawicki

Warsaw University of Technology

June 13, 2024

Contents

Part 1 (Transfer Learning for CNN and Transformers)

1 Datasets

1.1 DTD

Describable Textures Dataset used for training Transformers and CNN models written from scratch. Contains 5640 images of different textures organized according to 47 categories inspired from human perception:

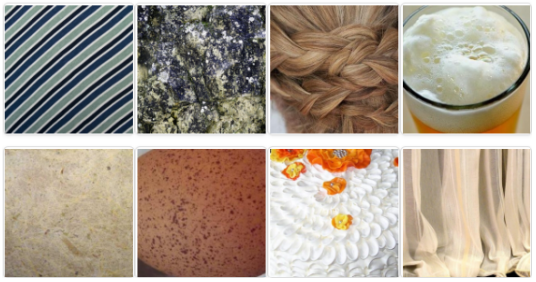

Figure 1: Describable Textures Dataset.

1.2 ImageNet

The models obtained from available sources had been pre-trained on ImageNet, a huge database of 14197122 images of 21841 categories.

1.3 RSSCN7

We applied transfer learning and curriculum learning to the RSSCN7 dataset, which consists of 2,800 satellite pictures divided into 7 classes, with each class containing 400 images.

Figure 2: RSSCN7.

1.3.1 DataLoader(Transformations of images in RSSNC7 dataset)

In our RSSNC7dataloader each image is resized to 256x256 pixels, converted to PyTorch tensors, and normalized. Additionally, the dataset was split into training and testing subsets, with 80% of the images designated for training and 20% for testing. While training of models we have set shuffle to True.

2 Introduction to part 1

We aimed to explore and compare different approaches to training deep learning models for image classification tasks. Our approach was structured as follows:

- **Training form scratch**. Transformers and CNN models were trained from scratch on DTD.
- **Models pre-trained on ImageNet**. We used available models of identical architecture but pre-trained on ImageNet.
- **Transfer learning**. To adapt models to the RSSCN7 we used transfer learning.
- **XAI**. To understand the behaviour of DL models, we used Explainable AI.

3 Transformers

We decided to perform training on a classical Vision Transformer model with 12 multi-head self attention blocks.

At the beginning, we took ViT model with default parameters. Afterwards, it was pretrained on DTD dataset. Due to the fact, that a part of photos had multiple classes, multi-label classification task was applied.

The training lasted for 40 epochs. In the meantime, the results were evaluated on both training and validation set.

The final results on test set gave 0.1177 loss and nearly 0.80 accuracy but here the prediction was treated as correct if it detected any of the image classes.

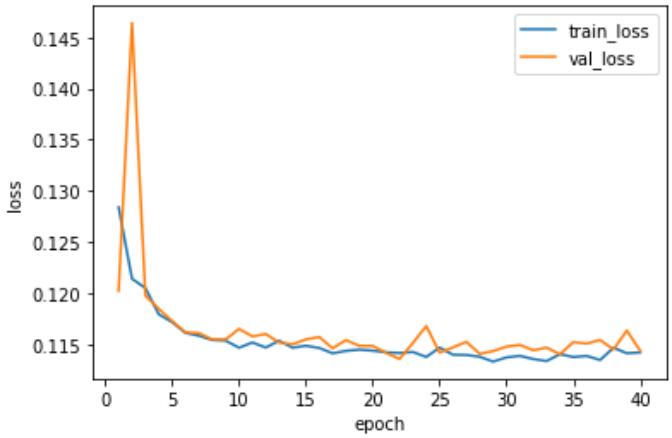

Figure 3: Loss results during pretraining ViT on DTD.

The next step was fine-tuning the model on RSSCN7 dataset. The training was conducted in a similar way. Number of epochs was also set to 40. Learning rate parameter had to be decreased to 0.0001. The final results are presented on a test dataset.

The loss was 1.78 and the accuracy score was 0.39.

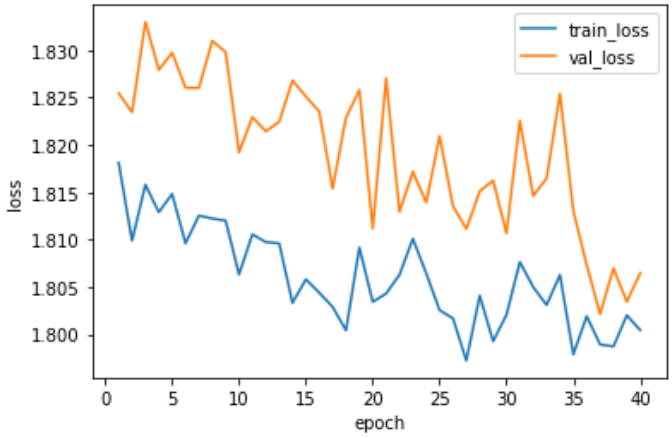

Figure 4: Loss results during fine-tuning ViT(DTD) on satellite photos.

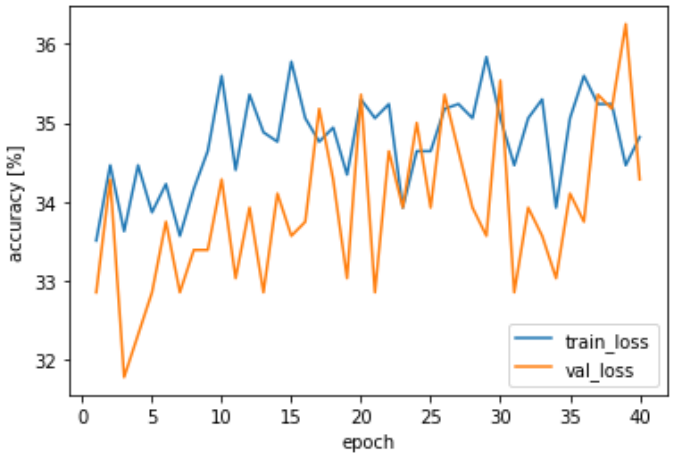

Figure 5: Accuracy scores during fine-tuning ViT(DTD) on satellite photos.

For comparison, we applied identical fine-tuning to ViT model pretrained on ImageNet. There were 40 epochs with learning rate set to 0.0005. The loss on training set was 1.93. The accuracy was 0.61.

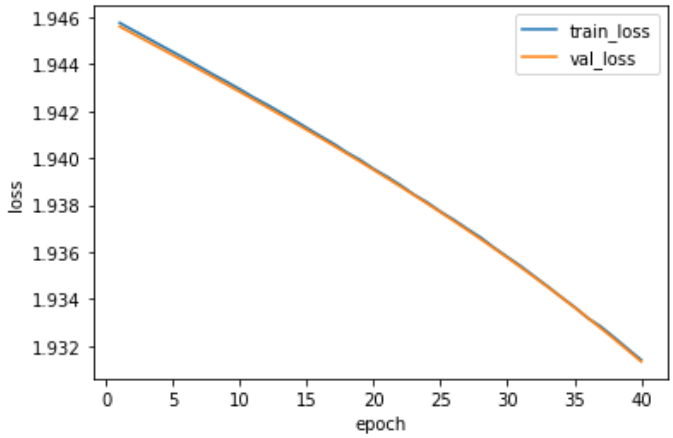

Figure 6: Loss results during fine-tuning ViT(ImN) on satellite photos.

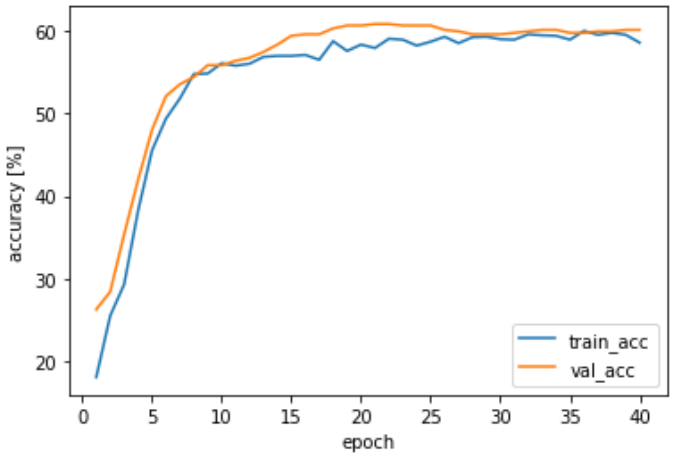

Figure 7: Accuracy scores during fine-tuning ViT(ImN) on satellite photos.

Both nets' final results are not satisfying. The training process was very time-consuming. It forced us to carry out the training on GPU. Probably, increasing training epochs could result in better scores. During fine-tuning, changing only the weights of the last layers could speed up the process.

Based on reached results, it is clear, that Vit(ImN) performed significantly better than ViT(DTD).

4 Convolutional Neural Networks

After carefully considering various convolutional neural network (CNN) models, including our own network, LeNet-5 and ResNet18, we settled on ResNet18. This choice was due to the optimal balance between power and compactness of the network. ResNet18 is powerful enough to handle our final dataset, yet compact enough to allow us to train it ourselves.

4.1 ResNet18 architecture

ResNet18 consists of 18 layers, including convolutional layers, pooling layers, and fully connected layer. It consists of 4 residual blocks. The architecture is shown below.

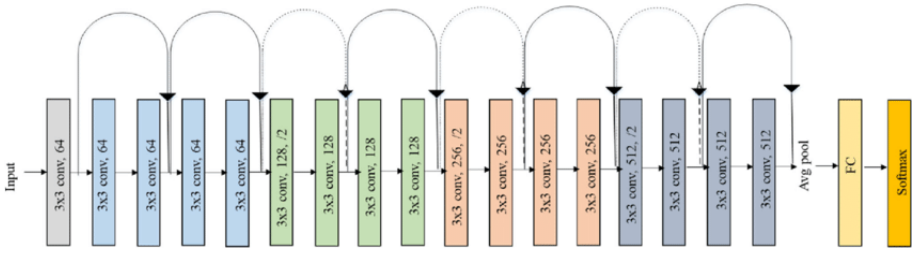

Figure 8: ResNet18 architecture

Among all transfer learning options, we selected the approach where all layers were frozen except for the the final layer (fully connected) of the neural networks.

4.2 The training process

Firstly, we pre-trained the model on the DTD dataset. We trained the model for 100 epochs, however the training process was time-consuming due to the fact that we have been training it on our laptops without 'cuda'. Similar to Transformers, the results were evaluated on test set as well. We divided the training process into parts: 0-50 epoch, 50-70 epoch, 70-80 epoch, 80-90 epoch.

The optimal model was the one trained for 90 epochs with best scores 0.33 loss and 0.91 accuracy on training set, and 2.41 loss and 0.52 accuracy on validation. Better accuracy might have been achieved with a better fine tuning of learning rates; however, our efforts were significantly constrained by the computational power of our available hardware, resulting in extremely long network training times.

We fine-tuned models on RSSCN7 dataset.We froze all the layers except the last, fully connected one, which was changed by us in order to deal with 7 classes. We trained model for 50 epochs and have not noticed any significant changes after 42th. The results were: 0.5297 loss and 85.5357% accuracy on test set.

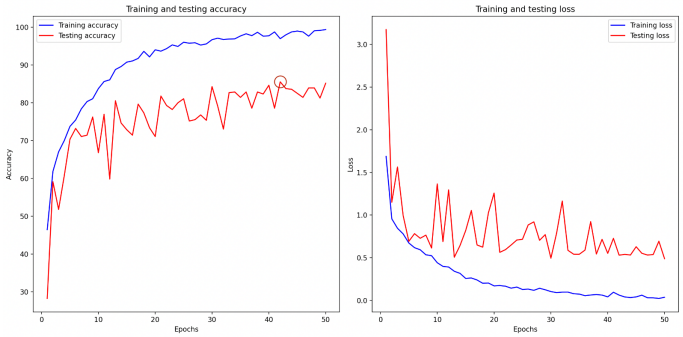

Figure 9: Accuracy and loss, model pre-trained on DTD.

Next we fine-tuned the model pre-trained on ImageNet. Similarly, this model was trained for 50 epochs. The optimal number of epochs was 23 with results: 0.4009 loss and 86.2500% accuracy on test set.

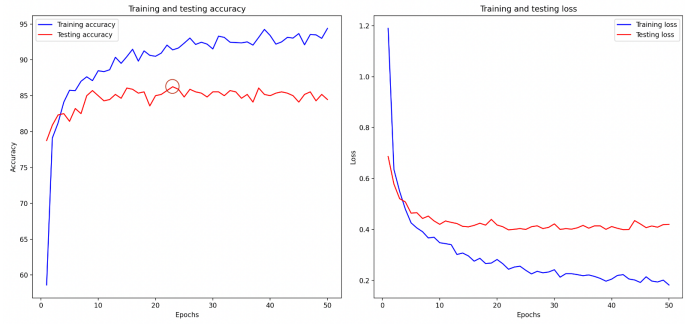

Figure 10: Accuracy and loss, model pre-trained on Imagenet.

Metrics values were close to each other, however the model pre-trained on ImageNet dataset had a bit better perfomance, was more stable.

5 XAI

To understand the behavior of deep learning models, we employed LIME. LIME helps to identify the most critical parts of images that contribute to correct class classification. By using LIME, we were able to illustrate how models make decisions, highlighting which regions of images were most influential in determining the correct outcome.

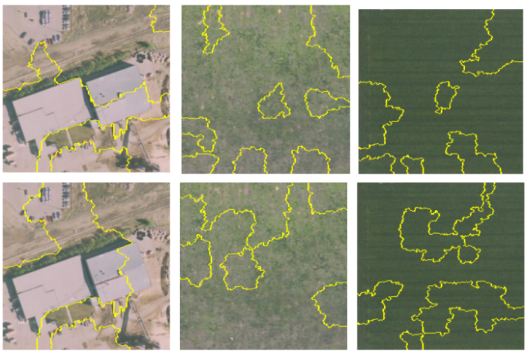

Figure 11: LIME. First line: pre-trained on DTD. Second line: pre-trained on ImageNet.

Figure 12: LIME. First line: pre-trained on DTD. Second line: pre-trained on ImageNet.

While the highlighted areas identified by LIME may not always match human perception, the model's choices, such as riverbanks and house contours, often seem reasonable.

PART 2 (2 Versions of Curriculum Learning for ResNet18)

6 Introduction to part 2

6.1 Our Motivation and hypotheses

In the second phase of our project, the aim was to enhance model performance by exploring alternative approaches to traditional transfer learning. We hypothesized whether curriculum learning could improve results.

Firstly, we decided to divide the images ourselves, but a question was how the human factor might affect the results. Specifically, we were concerned whether human involvement would decrease the performance. As a result we tested this hypothesis through two approaches:

- **Manual division of images**. We categorized the dataset into 3 subsets of increasing complexity.
- **Self-paced learning**. An automated method that dynamically adjusts the learning process based on the model's progress.

Additionally, we created a function to calculate the K-Nearest Neighbors (KNN) metric for CNN models. This metric allowed us to evaluate performance using not only accuracy and loss.

6.2 What is Curriculum Learning?

Curriculum Learning is a training strategy in machine learning where models get images while training in a meaningful order: it starts from getting easy images, after some epochs the complexity is increasing. This emulates the human learning processes, making the learning process more efficient and potentially improving the model's performance and generalization. By sorting the training data from easy to difficult, models can build a good foundation before coming across more challenging tasks.

7 Transfer Learning after selecting the best learning rates

We aimed to enhance the stability of our models by selecting appropriate hyperparameters. To achieve this, we manually adjusted the learning rates, changing it after reaching certain level of training accuracy, and trained models faster using Kaggle GPU. This way we improved the stability and performance of our models.

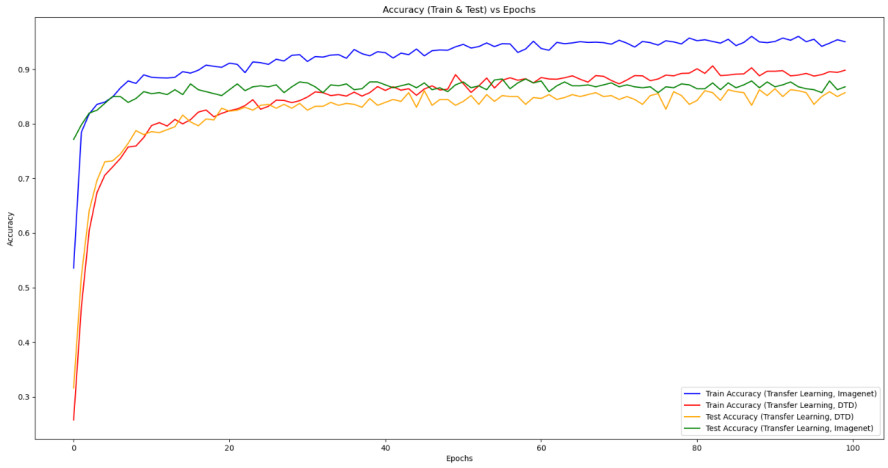

Figure 13: Accuracy of transfer learning.

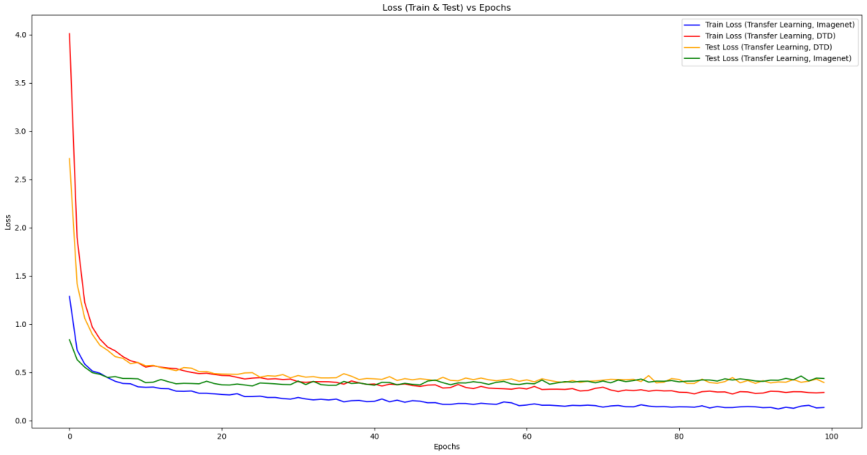

Figure 14: Loss of transfer learning.

We achieved performance improvements in our models:

- **Model pre-trained on DTD**. Loss decreased from 0.53 to 0.39, and accuracy improved from 0.85 to 0.86.
- **Model pre-trained on ImageNet**. Loss decreased from 0.40 to 0.38, and accuracy increased from 0.86 to 0.88.

The most significant outcome was the enhanced stability of our models. This stability is important for reliable model predictions.

8 Curriculum Learning (Manual Division)

Firstly, we manually divided the images into three folders: easy, medium and hard. The division per class is following:

- aGrass: easy - 135 items, medium - 166 items, hard - 99 items
- bField: easy - 136 items, medium - 164, hard - 100
- cIndustry: easy - 135 items, medium - 170, hard - 95
- dRiverLake: easy - 142 items, medium - 155, hard - 102
- eForest: easy - 142 items, medium - 156, hard - 100
- fResident: easy - 93 items, medium - 116, hard - 181
- gParking: easy - 125 items, medium - 138, hard - 135

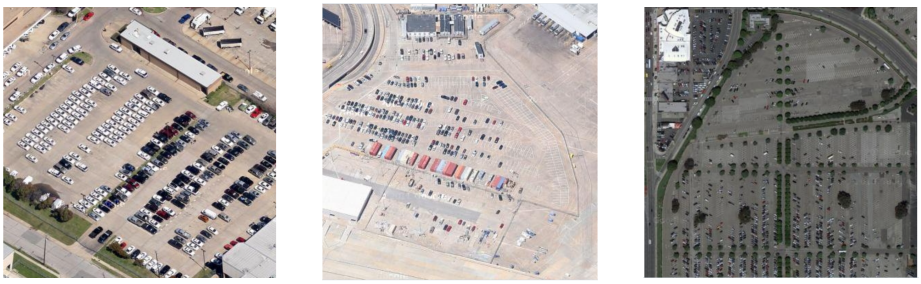

Figure 15: Parking easy, medium and hard.

8.1 Details in implementation

The model was trained in three stages using progressively challenging sets of images:

1. **Easy set**. Trained for 50 epochs on the easiest set of images.
2. **Medium set**. Trained for 80 epochs on easy merged with medium difficulty set.
3. **Hard set**. Trained for 20 epochs on all 3 sets of images.

This staged approach allowed the model to gradually adapt and improve its performance.

In our implementation:

1. The ResNet18(ImageNet) **model** was taken from torchvision.models with weights='ResNet18_Weights.DEFAULT'. The final fully connected layer was modified to output 7 classes.

2. **Batch size** was 32 suitable for the size of our dataset.
3. As a **Loss function** we have used nn.CrossEntropyLoss()
4. As an **Optimization algorithm** we have used optim.Adam()
5. The **Test set** always matched the difficulty level of the current training set.

8.2 Results

The final results on RSSCN7 dataset:

- Easy: accuracy = 0.91 and loss = 0.27.
- Easy and Medium: accuracy = 0.92 and loss = 0.25.
- Easy, Medium and Hard: accuracy = 0.94 and loss = 0.22.

The plots illustrating accuracy and loss are shown below.

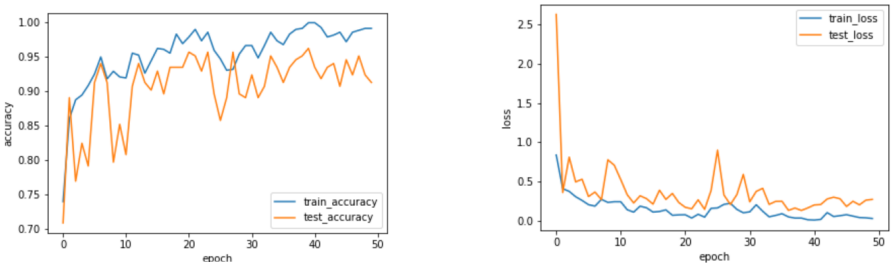

Figure 16: Easy: accuracy and loss.

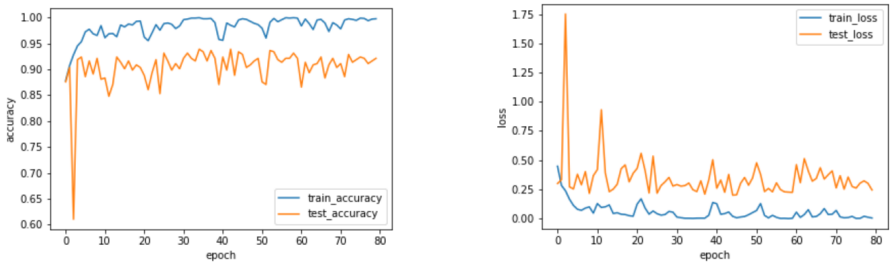

Figure 17: Medium: accuracy and loss.

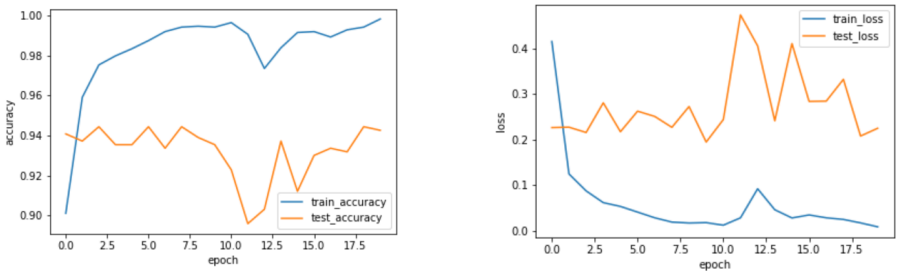

Figure 18: Hard: accuracy and loss.

9 Curriculum Learning (Self-Paced Learning)

9.1 General idea

We hypothesized that Curriculum Learning with manual division of images may not achieve the best results due to human error. Because of the non-human nature of the CNN model, a human and a model can assign coordinately different levels of difficulty to the same picture. Therefore, we decided to automatize the process of selecting appropriate images on certain stages of training . One of the methods in Curriculum learning that involve automatically selecting images is Self-Paced Learning Method.

In each epoch, the model sorts pictures from the entire available training set by difficulty, selects only a certain number ($x\%$) of the easiest ones, and trains only on them. Then the procedure is repeated, the value of x grows until it reaches 100%. Then the model is trained on all available training data until the best result is obtained.

9.2 Details in implementation

In our implementation:

1. The ResNet18(ImageNet) **model** was taken from torchvision.models with weights='ResNet18_Weights.DEFAULT'. The ResNet18(DTD)was trained by us in the 1 part of the project
2. **Batch size** was 32 due to not a big dataset
3. As a **Loss function** we have used nn.CrossEntropyLoss()
4. As an **Optimization algorithm** we have used optim.Adam()
5. We have used a **parameter** λ for defining the percent (number) of images that should be taken for training in a certain epoch. In the beginning of training $\lambda = 0.1 = 10\% = 224$ images.
6. The λ **parameter** was not increased in every epoch. The most effective way to increase lambda turned out to be this solution:

```
if train_accuracy > 0.85:
    if lambda_current < 0.6:
        lambda_current += step
        if lambda_current > 1:
            lambda_current = 1
    else:
        counter = counter + 1
        if counter % 2 == 0:
            lambda_current += step
            counter = 0
            if lambda_current > 1:
```

```
                    lambda_current = 1
```

In other words - the model had to reach 85% accuracy on the current training dataset before getting new 5%=112 (images) of the whole available training dataset. In addition, if the current training dataset has already reached 60% (let now be y%), then before getting a new portion of images, the model should have at least 2 times be trained on the images that were included in the training dataset of size y%.

7. We have been changing the **learning rate** during training. For different CNNs the best values of learning rates were different, but the general idea of the best approuch was to change learning rates depending on current train accuracy like this:

```
learning_rate = 0.001
....
def train_model(..):
    for epoch in range(num_epochs):
        ...
        if epoch_accuracy >= 85:
            learning_rate = 0.0005

        if epoch_accuracy >= 89:
            learning_rate = 0.00005

        if epoch_accuracy >= 92:
            learning_rate = 0.000001
        ...
```

8. **Test set** in order to evaluate the model on each epoch consisted of images of all levels of difficulty.

9. Curriculum learning does not necessitate the freezing of layers, and as such, we have not implemented this approach either.

10. In order to compare the time of training transfer learning models and self-paced learning models we have been training them on the same GPU on kaggle: GPU T4x2.

9.3 Results

We have applied the self-paced learning method to training of 2 CNNs: ResNet18 pretrained on DTD (trained by us, in the 1 part of the project) and ResNet18 pretrained on ImageNet. The plots of the loss function and accuracy are shown below.

It is worth noting that the network pretrained on DTD trained more slowly. This is undoubtedly due to the fact that the learning rates during the training of this model were lower than that of the model pretrained on ImageNet. However, we have chosen specificly these learning rates because at the end the network achieved the best results under exactly these conditions.

Based on the results obtained and the graphs below, it can be clearly stated that the ResNet18 pretrained on ImageNet performs better than the network trained on DTD.

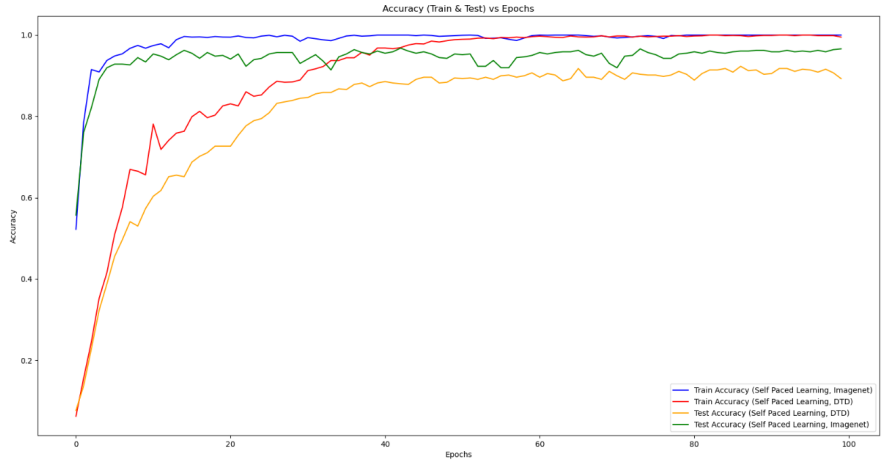

Figure 19: Accuracy of self-paced learning.

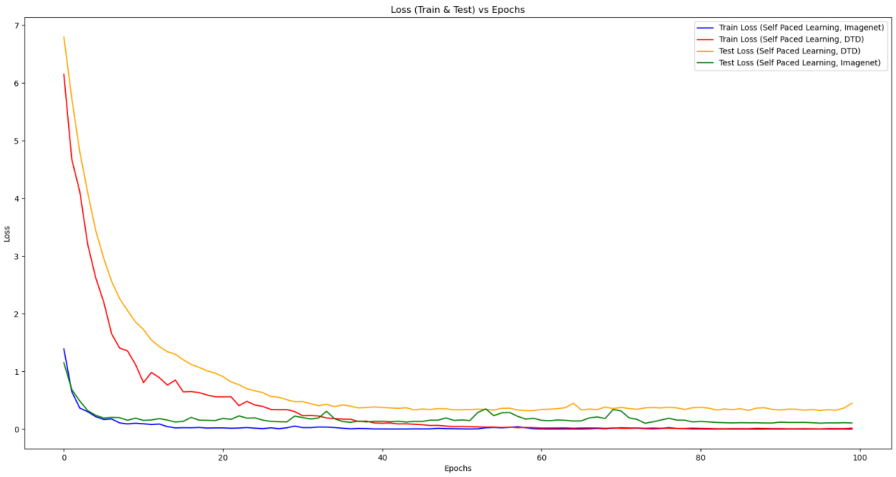

Figure 20: Loss of self-paced learning.

Some facts about self-paced learning of ResNet18 (pretrained on Imagenet)

(we heve chosen to tell about it, because it performs better):

1. The model was trained on the entire available training dataset for the first time during epoch 28.

2. Startling fact: At the 4th epoch, the model achieved 89% accuracy on the test set, taking 91 seconds to train on 20% of the whole available training dataset. This is already better than the best results achieved by ResNet18+Transfer Learning.

3. It took 43 epochs to achieve the best result, with an accuracy of 96.78% on the test set.

9.4 Analysis of Misclassification of Self-Paced (ImageNet)

The Self-Paced (ImageNet) model didn't reach 100 percent accuracy on test set, which prompted us to figure out which classes it most often struggled with and look at them. The graph of misclassifications and some unrecognized images are shown below.

The model demonstrates a 100% accuracy rate in recognizing images of forests, yet encounters more difficulties with images classified under the Parking category. A considerable portion of the misclassified images are those that present recognition challenges even for a human. This indicates that, while the model has successfully learned to identify the primary patterns associated with each class, it has not learned the patterns of atypical examples, so model generalizes well.

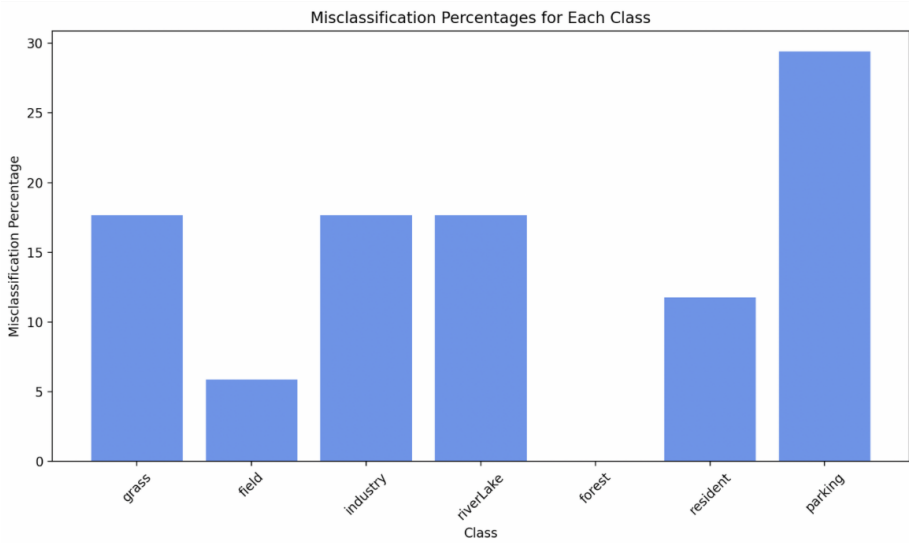

Figure 21: Misclassification percentage by class

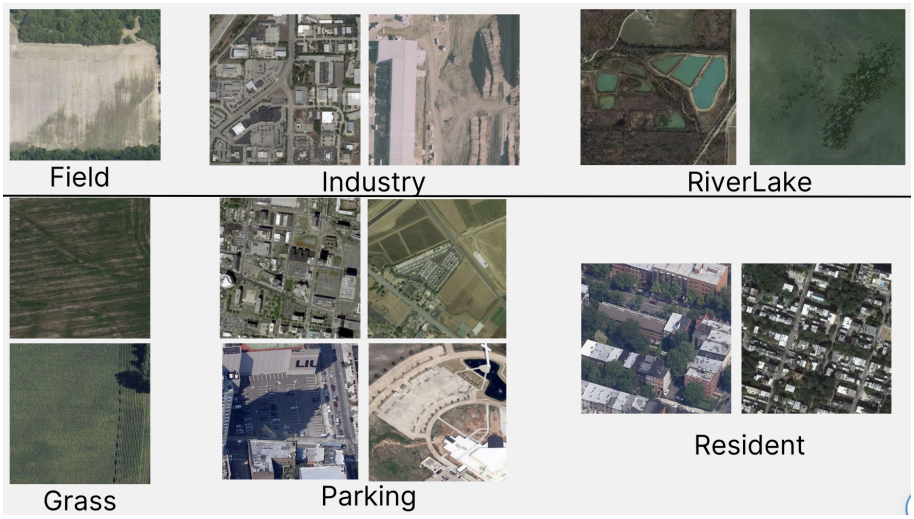

Figure 22: Images that were not correctly classified by The Self-Paced (ImageNet) model

9.5 Comparing training time of the best transfer learning model and of the best self-paced learning model

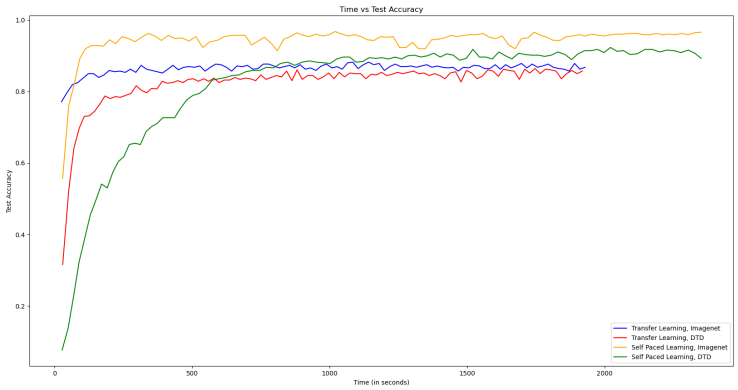

Figure 23: Test Accuracy vs Time (100 epochs)

Due to the fact that we trained the transfer learning models and self paced learning models on the same gpu on kaggle it makes sense to compare the time it took to train and the accuracy on the test set at this time.
In general, it is difficult to state anything about the time by looking at these time

graphs of our models because the self paced model immediately outperforms the other three.

10 KNN Metric

Our additional task was to introduce a custom metric. We called it KNN which stands for k nearest neighbors. The idea behind the metric was to uncover how a model represents pictures inside its architecture and how this representation is adapted during training process.

The metric is count on a given dataset. All the pictures from it are passed to a model. Our metric extract the output of penultimate layer for every picture. This way it receives a set of tensors. Each tensor is then flattened. In the next step, we iterate over tensors and count distances between selected tensor and all others. This way only k nearest tensors are selected for each tensor in the set. Afterwards, the original labels of the tenors are compared. For every tensor, we count what fraction of its neighbors have the same label. Finally, the the results are aggregated using an average.

11 KNN Metric comparison

In our experiments, the k in KNN metric was set to 5. We observed interesting behaviors in the KNN metric across different training methodologies:

- **Transfer learning**. The KNN metric showed stability, not growing much throughout epochs.
- **Self-paced learning**. The KNN metric quickly achieved high values and maintaining stability throughout epochs.
- **Curriculum learning (Manual division)**. The KNN metric had high starting value and showed limited growth over epochs.

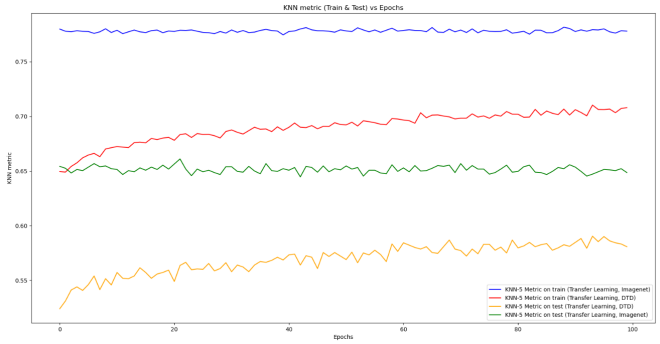

Figure 24: KNN of transfer learning.

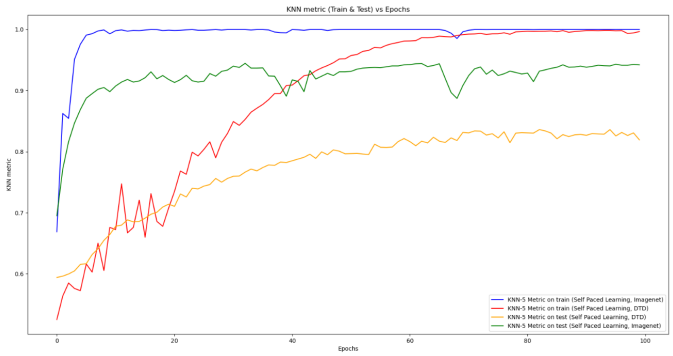

Figure 25: KNN of self-paced learning.

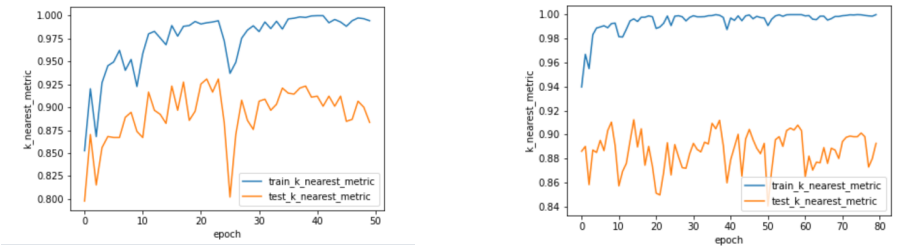

Figure 26: KNN of manual division curriculum learning: easy and easy with medium.

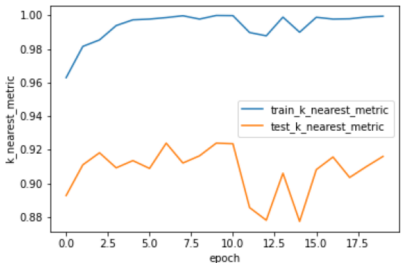

Figure 27: KNN of manual division curriculum learning: on all 3 sets.

12 Final results and Conclusion

In conclusion, our experiment confirmed that curriculum learning significantly boosted performance across accuracy, loss, and the K-Nearest Neighbors metric. We observed that self-paced learning outperformed manual division, establishing it as the superior automated approach. In contrast, transfer learning

(1 version chosen by us) showed poorer results, indicating the advantages of training models on progressively challenging samples.

Notably, the best-performing model overall was the Self-Paced learning approach applied to a model pre-trained on ImageNet.

Our work highlights the importance of strategic model training and the effectiveness of curriculum learning in enhancing model performance.

	Transfer Learning (ImageNet)	Transfer Learning (DTD)	Manual Division (ImageNet)	Self-Paced Learning (ImageNet)	Self-Paced Learning (DTD)
Accuracy	0.88	0.86	0.94	0.97	0.92
Loss	0.38	0.39	0.22	0.13	0.32
KNN	0.66	0.59	0.92	0.94	0.84

Table 1: Metrics of trained model.

13 Contribution of members

Jakub Sawicki	- Implementation of transformer training scripts: pretraining on DTD and fine-tuning on RSSCN7. - Pretrainig ViT model on DTD. - Implementation of KNN metric. - Annotating RSSCN7 classes d and e. - Creating script for curriculum learning (manual division). - Training CNN on easy and easy with added medium sets for curriculum learning.
Rafał Pyzowski	- Implementation of transformer training scripts: pretraining on DTD and fine-tuning on RSSCN7. - Fine-tuning ViT model on RSSCN7. - XAI for transformers. - Annotating RSSCN7 classes f and g. - Training CNN on full dataset for curriculum learning.
Milanna Pahasian	- Implementation of training process and training of: ResNet18 from 0 on DTD, ResNet18(pretrained on DTD + Transfer Learning) on RSSCN7, ResNet18(pretrained on ImageNet + Transfer Learning) on RSSCN7. - Adding and computing K-Nearest Neighbors metric of ResNet18(pretrained on DTD + Transfer Learning), ResNet18(pretrained on ImageNet + Transfer Learning), ResNet18(pretrained on DTD + Self-Paced Learning), ResNet18(pretrained on ImageNet + Self-Paced Learning). - Manual division of images (class c and half of class b) from RSSCN7 dataset for Curriculum Learning.
Katsiaryna Bokhan	- Implementation of training scripts and training (with hyperparameters tuning) of: ResNet18 from 0 on DTD, ResNet18(pretrained on DTD + Transfer Learning) on RSSCN7, ResNet18(pretrained on ImageNet + Transfer Learning) on RSSCN7. During these trainings KNN metric was not counted. - Implementation of training scripts and training (with hyperparameters tuning) of: ResNet18(pretrained on DTD + Self-Paced Learning) on RSSCN7, ResNet18(pretrained on ImageNet + Self-Paced Learning) on RSSCN7. During these trainings KNN metric was not counted. - Manual division of images (class a and half of class b) from RSSCN7 dataset for Curriculum Learning. - Plotting result graphs of 5 previuos CNNs. - Applying XAI(LIME) to 2 Transfer Learning CNNs. - Analysis of Misclassification of Self-Paced (ImageNet)

References

1. RSSCN7 dataset. https://github.com/palewithout/RSSCN7
2. Describable Textures Dataset (DTD). https://www.robots.ox.ac.uk/ vgg/data/dtd/
3. ImageNet dataset. https://www.image-net.org/
4. The GitHub repository, an inspiration for the first part of the project (CNN). https://github.com/sakshamsds/describable-textures-dataset-classification/tree/main
5. The GitHub repository with works on the topic of curriculum learning. https://github.com/Openning07/awesome-curriculum-learning?tab=readme-ov-file
6. Bengio, Y., Louradour, J., Collobert, R., Weston, J. (2009). Curriculum Learning. https://www.researchgate.net/publication/221344862_Curriculum_learning
7. Jiang, L., Meng, D., Zhao, Q., Shan, S., Hauptmann, A. G. (2015). Self-Paced Curriculum Learning. http://www.lujiang.info/camera_ready_papers/AAAI_SPCL_2015.pdf