Plant Seedlings Classification

Kristi Topollai
Electrical Engineering
New York University
New York, NY 10012
kt2664@nyu.edu

Qiwen Zhang
Center for Data Science
New York University
New York, NY 10012
qz2274@nyu.edu

Lehan Li
Center for Data Science
New York University
New York, NY 10012
ll3745@nyu.edu

Abstract

The project aims to classify different types of plant seedings based on images. The project uses various image prepossessing methods including image transformation, augmentation, segmentation and smote.Various deep learning models are employed including ResNet 50, ResNet 101, Inception_v3 and Data-Efficient Image Transformers. Different training techniques are used such as Snapshot Ensembles and Optuna hyperparameter search. The project achieved a test micro-averaged F1-score of 0.98362 [1].

1 Introduction

Computer Vision has been widely used in the field of agricultural and environment science. The project aims to build a computer vision model that could detect weed of a various types of crop seedling. The dataset is obtained from kaggle competition ”Plant Seedlings Classification” based on the open source dataset from The Aarhus University Signal Processing group and University of Southern Denmark. The dataset contains 12 species and a total of approximately 960 unique plants' images. The project contains various image preprocessing techniques and explores multiple deep learning models to achieve a good performance.

2 Data Preprocessing

Data preprocessing is essential for deep learning model's performance. The aim of data preprocessing is to add variability into training data so that the model could have a better performance for unseen dataset. Moreover, preprocessing also helps to extract useful information in the image before feeding the input into the model. And finally, preprocessing could help to alleviate class imbalance issue. Total of four types of image preprocessing techniques are conducted in the project: Transformation and Augmentation, Image Segmentation, Vanilla Gan Model and SMOTE.

2.1 Transformation and Augmentation

Traditional transformation technique is employed including

- rotation of 180 degree
- random affine transformation
- vertical and horizontal flip
- addition of Gaussin random noise

[1]Some parts of the report were writtern with the assistance of ChatGPT. Code is available at https://github.com/QiwenZz/Ds-ga-3001-Statistical-learning-project

- randomly change the brightness, contrast, saturation and hue of an image
- resize of the image
- image standardization

These transformations are basic techniques used to add randomness into the dataset. And except image standardization, these techniques are independent of the content / pixel of the original images.

Besides basic transformations, random augmentation and auto augmentation are employed. Auto Augmentation[3] is a technique that ”automatically search for improved data augmentation policies”. It designed a search space where a policy consists of many subpolicies, one of which is randomly chosen for each image in each mini-batch.” And random augmentation [4] is a technique which significantly reduce the search space of policy and shorten the computational time for preprocessing time.

2.2 Image Segmentation

Image segmentation is a technique that segment the target object in the image while converting the background color of the image into black. It could help to capture the shape of the plants and emphasize the target object before feeding the data into the model. The entire operation is based on opencv module.

Image segmentation consists of three steps: masking, segmentation and sharpening. During the masking phrase, the model creates the mask based on the HSV color space o detect the color of the object. Then we apply morphological operation to remove the distortion created by noise and texture. Afterwards, the project segments the object based on the mask result from the first phrase. The final stage is to use Gaussian Blur which convolves the image with a Gaussian low-pass filter in order to removes / removes the high-frequency components and blur the image [Figure 1].

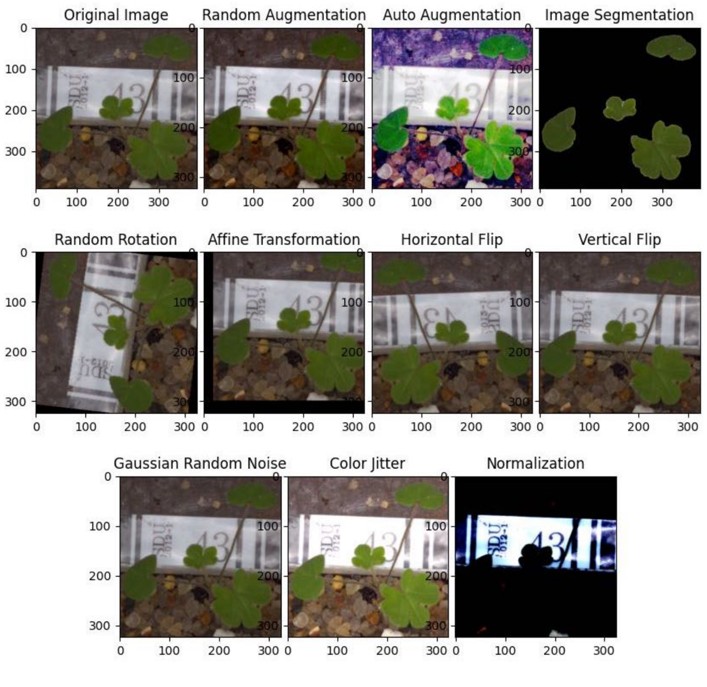

Figure 1: Augmentations

2.3 Vanilla Gan Model

Since the database contains a limited number of data points, Vanilla Gan is used to expand the training data by generating artificial training images. Vanilla Gan is trained from scratch using the existing training data. However, Vanilla Gan cannot captures the precious texture of the plant as well as the detailed shape for the leaves. Instead, the generated images only capture the rocky background with a few green pixels in the center of the images.

Due to the unsatisfied performance of Vanilla Gan, the project did not use any of the artificial images in the model training process. Instead, the project employs transformations and segmentation on the original dataset as the image preprocessing stage.

2.4 SMOTE

Synthetic Minority Oversampling TEchnique, [2] or SMOTE for short, is a widely used technique to synthesize new examples for tackling the issue of class imbalance. It generates synthetic samples for the minority class by interpolating new instances between existing ones. This technique helps to balance the class distribution and mitigate the potential bias towards the majority class during model training. It works by first randomly selecting a sample from a minority class and then identify k nearest neighbors using the KNN algorithm. Then it takes one of the neighbors and identify the vector between the original point and the neighbor. A random number between 0 and 1 is multiplied to this vector and a synthetic data point is created.

We applied SMOTE to oversample the minority class and also randomly downsample the majority class at the same time, thereby creating synthetic examples that are similar to the existing minority class instances and making every class have the same number of training examples. By incorporating SMOTE, we aimed to enhance the generalization capability of our classification model and improve its performance in accurately predicting both minority and majority class.

3 Model

3.1 Pretrained Models

3.1.1 ResNet

ResNet, short for Residual Network, is a deep neural network architecture that was introduced by Microsoft Research in 2015 [7]. The main idea behind ResNet is to tackle the problem of vanishing gradients in very deep neural networks. The vanishing gradients problem occurs when gradients become too small during backpropagation, making it difficult for the network to learn from earlier layers. To address this issue, ResNet introduces skip connections, which allow information from earlier layers to bypass several layers and reach deeper layers. This way, the network can learn from the earlier layers while still making progress in the deeper layers. ResNet also uses a convolutional layer with a kernel size of 1 to reduce the dimensionality of the feature maps before each skip connection.

ResNet comes in different variants, such as ResNet18, ResNet50, ResNet101, and ResNet152, which differ in their depth and number of parameters. For example, ResNet18 has 18 layers and approximately 11 million parameters, while ResNet50 has 50 layers and approximately 23 million parameters. ResNet50 is often used as a benchmark for image classification tasks, and it has achieved state-of-the-art performance on various datasets, including ImageNet. ResNet has also been successfully applied to other computer vision tasks, such as object detection, semantic segmentation, and image captioning.

3.1.2 Inception

Inception is a deep neural network architecture that was introduced by Google in 2014 [11]. The main idea behind Inception is to improve the efficiency and accuracy of deep neural networks by using a combination of different convolutional filters. Instead of using only one size of convolutional filters, Inception uses filters of different sizes, such as 1x1, 3x3, and 5x5, in parallel. This allows the network to capture both fine-grained and large-scale features, which can be important for image classification tasks. Inception also uses a technique called dimensionality reduction, which reduces the number of features before each convolutional layer to reduce the computational cost of the network.

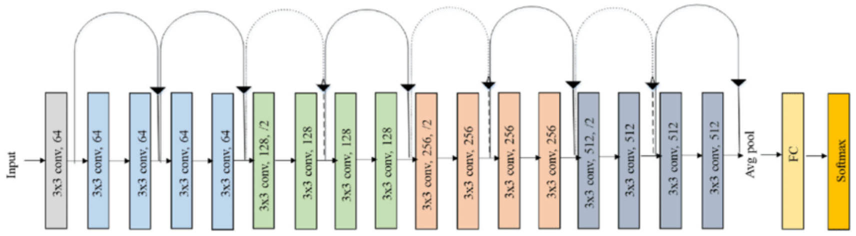

Figure 2: Resnet18 architecture

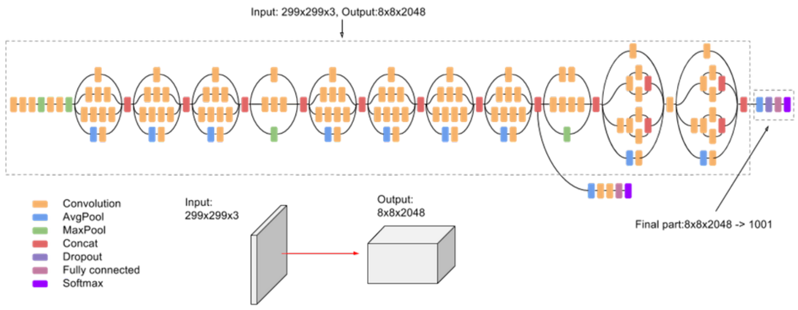

Figure 3: Inception v3 architecture

Like ResNet, Inception comes in different variants, such as InceptionV1, InceptionV2, and InceptionV3, which differ in their architecture and number of parameters. InceptionV3, for example, has 48 layers and approximately 23 million parameters. Inception has achieved state-of-the-art performance on various computer vision tasks, including image classification, object detection, and semantic segmentation. Inception has also been used in other domains, such as natural language processing and speech recognition, where it has shown promising results. Inception's ability to capture both fine-grained and large-scale features, combined with its computational efficiency, makes it a popular choice for many deep learning applications.

3.1.3 DeiT

DeiT[12], which stands for ”Data-efficient image Transformers,” is a deep learning model for image classification tasks. It leverages the Transformer architecture, and applies it to the field of computer vision. Unlike traditional convolutional neural networks (CNNs), which rely on hand-designed hierarchical feature extraction, DeiT adopts a self-attention mechanism to capture global dependencies in the image.

At its core, DeiT consists of a stack of Transformer encoder layers, where each layer consists of multi-head self-attention and feed-forward neural network modules. The self-attention mechanism allows the model to attend to different parts of the image when making predictions, enabling it to capture long-range dependencies and spatial relationships effectively. Additionally, DeiT utilizes positional embeddings to encode the spatial information of the input image.

Models within the vision transformer family generally requires a large amount of training data, DeiT, however, introduces an extra distillation token on the top of a basic vision transformer. This token passes its output from attention layers into another strong image classifier, which could be a convnet, and uses this classifier as the teacher to distill the knowledge it learned to a small and compact student model. In this way, the number of data for training is substantially decreased compared to a ViT model. Given the limited number of data we have, DeiT might be the most appropriate vision transformer variant that we could test.

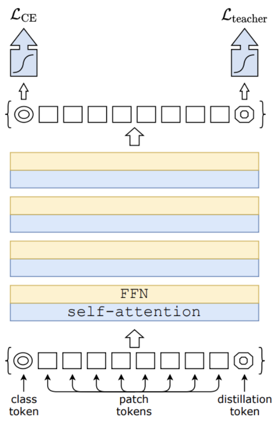

Figure 4: DeiT architecture

3.2 Finetuning

Fine-tuning is a transfer learning technique in which a pre-trained model is further trained on a new dataset to perform a specific task. The pre-trained model is usually trained on a large dataset, typically with millions of images or texts, to learn general features such as edges, shapes, or semantic meanings. Fine-tuning aims to reuse these general features learned by the pre-trained model and adapt them to the new dataset, which typically has fewer samples than the pre-training dataset.

In fine-tuning, the pre-trained model's weights are initially frozen, except for the last layer that is connected to the new dataset's output. This last layer is randomly initialized, and only its weights are updated during training, while the rest of the network remains frozen. The new dataset is then used to train the model, optimizing the last layer's weights to match the new dataset's labels.

Once the fine-tuning on the new dataset is complete, the model can be further fine-tuned by gradually unfreezing the earlier layers of the pre-trained model and continuing the training with a smaller learning rate. This step can be useful when the new dataset is larger or different from the pre-training dataset.

The fine-tuning process can be performed on various pre-trained models, including the aforementioned ones. Instead of the standard approach of freezing all the feature layers and only training on the classification fully connected head, we instead unfreeze some of the final feature layers and skip the finetuning part altogether.

In the case of the Resnets, we unfreeze one or two residual blocks, while for the Inception v3 we also unfreeze the final inception block. For DeiT the approach is...

Finally, as suggested by the competition's discussion, we also tried to finetune our trained models on the same dataset, but of different input size. Specifically, once we train our pretrained model on our dataset for a specific image size, we increase the image/input size and finetune that trained model. The idea behind this approach is to more efficiently force the network to first learn the low level features before proceeding with the high level ones. Unfortunately this approach was not very successful and will not be further discussed.

4 Optimization and Ensembling

4.1 Ensembling

Ensembling [6] is a technique used in machine learning and deep learning to improve the predictive performance of a model by combining the outputs of multiple models. The basic idea behind ensembling is that by combining the predictions of multiple models, we can reduce the overall prediction error and improve the overall accuracy of the predictions.

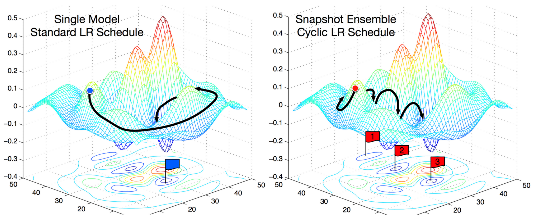

Figure 5: Snapshot ensembling

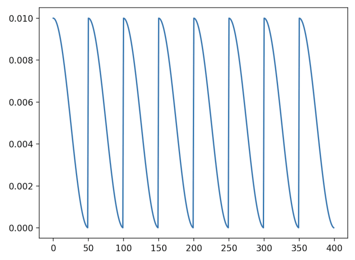

Figure 6: Cyclic LR

It is commonly used in competitions, where the best-performing models are often ensembles of multiple models. However, ensembling can also be computationally expensive and requires additional efforts to train and maintain multiple models.

4.1.1 Stacking

In stacking, multiple models are trained independently, and their predictions are used as input features to train a meta-model that produces the final prediction. Stacking allows for more complex relationships between the models and the data and can often lead to better predictive performance. In our case, instead of using the model outputs as features to a different model, we instead use them as they are by either averaging the logits or picking the output class for an input by majority voting.

In particular, we conduct different experiments with different hyperparameters and use ensembling on all these models. However this approach requires training many individual models.

4.1.2 Snapshot Ensembeling

The Snapshot Ensembling [8] technique involves the use of a cyclic learning rate schedule during training, where the learning rate is increased and decreased in a cyclical fashion. During training, the model is periodically saved, with each saved model having a different set of weights and hyperparameters.

The idea is that, when the learning rate is decreasing, the model converges to a local minimum. Once the cycle is over, this local minimum which is described by the model parameters at that time step is saved, on the new cycle, the learning rate is reset to a high value, which in turn forces the optimizer to exit this local minimum and search for a new one. The different local minima (different models) are then ensembled to produce the final output.

The hyperparameters for this method are the number of estimators, which is the desired number of local minima, and the total number of epochs. Using these two we can calculate the cycle duration and define the cycling learning rate schedule.

4.2 Optimizer and learning rate scheduling

4.2.1 Optimizer

For the optimization we follow a standard approach and use the usual methods that have been shown to perform well on a variety of tasks. We tried Momentum SGD, Adam [9], and weight decay as the regularizer.

Momentum SGD is an extension of stochastic gradient descent (SGD) that introduces a momentum term that accelerates the gradient descent along directions that consistently point in the same direction, thereby enabling faster convergence. Adam, on the other hand, is an adaptive learning rate optimization algorithm that combines the advantages of adaptive gradient algorithms and momentum. It adjusts the learning rate for each parameter based on the historical gradient information and dynamically adapts to the geometry of the loss surface. Weight decay is a regularization technique that adds a penalty term to the loss function proportional to the magnitude of the weights, which discourages large weights and helps prevent overfitting. Specifically, we use ADAM when we don't snapshot ensembling and momentum SGD when we do.

4.2.2 Learning rate schedule

In addition, when not using snapshot ensembling, we also use learning rate scheduling, Learning rate scheduling is a technique used in deep learning to adjust the learning rate during training. The goal of learning rate scheduling is to help the optimizer converge more quickly to a good solution and avoid getting stuck in a suboptimal local minimum or plateau.

Specifically we use cosine annealing with warm restarts [10], a scheduler similar to the cyclic schedule used for the snapshot ensembling. In cosine annealing, the learning rate is decreased gradually over a certain number of epochs, following a cosine function. This helps the optimizer converge to a better solution by gradually reducing the learning rate as the model gets closer to the optimal solution.

Warm restarts are used to help the optimizer escape from local minima and reach better solutions. In warm restarts, the learning rate is increased to a higher value after a certain number of epochs, which causes the optimizer to jump out of the current local minimum and explore other regions of the solution space. After the restart, the learning rate is again decreased according to the cosine annealing schedule.

Finally we also perform early stopping. Early stopping involves monitoring the performance of the model on a validation set during training and stopping the training process when the performance on the validation set stops improving or starts to degrade. This helps to prevent the model from becoming too complex and overfitting the training data, leading to better generalization and improved performance on unseen data. A similar approach is to save the model with the best performance on the validation set during training, however the advantage of early stopping over this approach is that it does not require for all epochs to be performed, something that saves a lot of time.

4.3 Hyperparameter Searching with Optuna

To search for optimal hyperparameters in our experiments, we employed Optuna [1], an open-source software library designed for efficient hyperparameter tuning. Optuna uses a combination of Bayesian optimization and pruning techniques to intelligently explore the hyperparameter search space and find the best set of hyperparameters for our model.

The core concept behind Optuna is to define an objective function that evaluates the performance of a given set of hyperparameters. Optuna then iteratively explores the hyperparameter space by sampling different configurations and observing their corresponding objective function values. Based on these observations, Optuna employs Bayesian optimization to construct a probabilistic model of the objective function and uses it to suggest promising areas in the search space.

Additionally, Optuna incorporates pruning techniques to early terminate unpromising trials during the search process. This approach saves computational resources by dynamically monitoring the intermediate results of trials and stopping those that are unlikely to yield better performance.

In our experiments, we defined the search space of hyperparameters, including learning rate, weight decay, and momentum, and specified the objective function that maximizes the accuracy on the validaiton data. Optuna then automatically explored this search space, iteratively sampling and evaluating different hyperparameter configurations until the maximum number of trials or a predefined stopping criterion was reached.

However, due to the constrained computation power, running even one trial of hyperparameter searching using optuna requires more time than we expected. Hence, no results were obtained using this package.

5 Results

For image preprocessing, we find out that the three preprocessing techniques (Auto Augmentation, Random Augmentation and Image Segmentation) works well when they are used along. And image segmentation technique leads to the best performance among three preprocessing techniques. However, the best performed model is trained using the original dataset. Moreover, we find out that it is useful to turn up the brightness of the image since it adds more contrast between the plant and the background rock when training with original images.

We tried various combination of deep learning models with image preprocessing techniques, some of the results are shown in the below table. The most successful ensembling method was snapshot ensembling.

We first used a train validation data split to find the method that best performs on the validation set without overfitting, after that we train on the whole dataset. Interestingly, combining all different augmentation methods did not yield the best results, specifically the best results were obtained when using simple augmentation instead of the more involved ones such as AutoAugment, Random augmentation and segmentation.

The best performing model was convincingly Resnet50, the more complex architectures such as deeper resnets, DeiT and Inception did not perform as well. One reason for that is the small size of the training dataset.

Hyperparameter	Value
Preprocessing	
SMOTE	True
RandAugment	False
AutoAugment	False
Segmentation	False
Size	324x324
Normalization mean	[0.485,0.456,0.406]
Normalization std	[0.229,0.224,0.225]
Brightness	0.8-2.0
Noise std	0.05
Model	
Architecture	Resnet50
Blocks to freeze	7
Training	
Optimizer	SGD-M
Learning rate	0.1
Momentum parameter	0.9
Scheduler	cyclic
Batch size	64
Epochs	200
Ensembling	
Ensemble method	Snapshot Ensembling
Estimators	10
Voting	average

Table 1: Hyperparameters for best performing model, F1 Score: 0.983

Model	Image Preprocessing	F1 Score
Snapshot Ensembeling	Auto Augmentation	0.956
Snapshot Ensembeling	Rand Augmentation	0.961
Snapshot Ensembeling	Image Segmentation	0.974
Snapshot Ensembeling	Basic Augmentation	0.983

Table 2: Different augmentation schemes and results

Applying SMOTE	Average Accuracy on Validation Set
True	0.9694
False	0.972

Table 3: Effect of the employment of SMOTE

6 Discussion

ResNet 50 generally has a better performance compared to DeiT. This could be attributed to the limited amount of trainng samples. The largest difference between two models is that ResNet50 is a typical CNN network with residual connections whereas DeiT is a variation of vision transformer which does not assume any inductive bias and uses attention to extract features. In the original ViT paper, ResNet 50 exceeds the performance of any ViT models until 100 million samples are given for pretraining [5]. While DeiT applies a distillation token to let a pre-trained classification model distill knowledge for it to learn, which reduces the number of samples needed, its architecture is still essentially a vision transformer, which can be impacted by a limited number of training data given.

Auto augmentation, random augmentation and image segmentation all results in a relatively lower f1 score compared to basic augmentations as shown in table 2. The reason why random augmentation did not perform well might be due to its huge transformation search space, which might contain transformations that actually hurt this weed/crop seedling classification problem. For example, shape could be a strong feature that differentiate weed/crop seedling. However, if a transformation such as Solarize is picked, it will damage the original shape that the plant has which makes the performance worse. The result of auto augmentation shows that it has the worst f1 score among all augmentation techniques. Such result is reasonable as we applied augmentation policy learned from cifar10, imagenet and svhn. It might not be appropriate to apply these learned policies directly given the different characteristics of the data. Our data mainly focuses on weed and seedling images where almost all images have similar background and only the plant part is the useful features. In dataset such as cifar10 and imagenet, background information is much more complex and the classification target is much more diverse. Also, aspect ratio differs from dataset to dataset, we need to accommodate these differences for our data. Hence, augmentation policies learned from dataset such as cifar10 might not be directly adaptable for our problem. Image segmentation, on the other hand, gives the higher f1 score compared to auto and random augmentation, but is still lower than applying basic augmentation. This could be due to the fact that while image segmentation reduces noises and can make our classificaiton model easier to grab useful features, it can result in information loss on the existing data. In the future, we could attempt to train one model based on the segmented data and one model on the original data and combine these two models using some strategies such as pooling the feature map in the last layer to combine the advantages that two different dataset bring.

One interesting finding is that after applying SMOTE to our dataset, the average accuracy of classification decreases. The original data is imbalanced where the smallest group has 221 samples and the largest group has 654 samples, which could make the model overfit on the minor groups. After applying SMOTE, each group has the same number of samples of 523, which tackles the problem of class imbalance. However, applying SMOTE does not improve the accuracy on validation set, this might be due to that firstly SMOTE is not approapriate for our image dataset. SMOTE works by finding a point's neighbor and randomly times a random number between 0 and 1 to the vector of the difference between the point and its neighbor to generate a new point. This might not work on our data because each point in our dataset is an image, which is high dimensional and complex (3x324x324). A nearest point is calculated using euclidean distance but it might not be an accurate way to find a neighbor that's similar to the original point in such high dimensional and complex feature space. Also, even the neighbor is similar, multiply the difference between these two image tensor by a random number of 0 to 1 might introduce bias and dissertation to the new synthetic point. Secondly, accuracy might not be a good indicator of SMOTE's effect in the first place. SMOTE prevents overfitting by generating synthetic examples for minor classes which can also hurt its predictive ability on the major classes. Using accuracy as the metric might underestimate the model's ability of classifying minor classes correctly. F1 score might be a better metric to compare models' performances on every class and assess SMOTE's effect in the current dataset.

In the future, for image preprocessing, we could try to combine the model's final layers trained by original image with the model trained by segmented image in order to enlarge the training dataset and better capture different textures and details. If we have enough time or computation resource, we can also try applying Bayesian optimization method to find the best set of hyperparameters.

References

[1] Takuya Akiba et al. *Optuna: A Next-generation Hyperparameter Optimization Framework*. 2019. arXiv: `1907.10902 [cs.LG]`.

[2] N. V. Chawla et al. "SMOTE: Synthetic Minority Over-sampling Technique". In: *Journal of Artificial Intelligence Research* 16 (June 2002), pp. 321–357. DOI: `10.1613/jair.953`. URL: `https://doi.org/10.1613%5C%2Fjair.953`.

[3] Ekin D. Cubuk et al. *AutoAugment: Learning Augmentation Policies from Data*. 2019. arXiv: `1805.09501 [cs.CV]`.

[4] Ekin D. Cubuk et al. "RandAugment: Practical data augmentation with no separate search". In: *CoRR* abs/1909.13719 (2019). arXiv: `1909.13719`. URL: `http://arxiv.org/abs/1909.13719`.

[5] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: `2010.11929 [cs.CV]`.

[6] M.A. Ganaie et al. "Ensemble deep learning: A review". In: *Engineering Applications of Artificial Intelligence* 115 (Oct. 2022), p. 105151. DOI: `10.1016/j.engappai.2022.105151`. URL: `https://doi.org/10.1016%2Fj.engappai.2022.105151`.

[7] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: `1512.03385 [cs.CV]`.

[8] Gao Huang et al. *Snapshot Ensembles: Train 1, get M for free*. 2017. arXiv: `1704.00109 [cs.LG]`.

[9] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: `1412.6980 [cs.LG]`.

[10] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. 2017. arXiv: `1608.03983 [cs.LG]`.

[11] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: `1409.4842 [cs.CV]`.

[12] Hugo Touvron et al. *Training data-efficient image transformers distillation through attention*. 2021. arXiv: `2012.12877 [cs.CV]`.