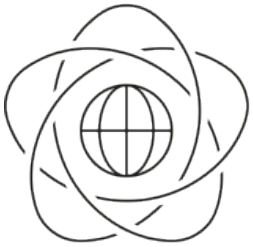

Wydział Matematyki
i Nauk Informacyjnych

Politechnika Warszawska

Deep Learning, Deep Learning Methods

Project 1: CNN

Mikołaj Rzepiński, Krzysztof Wolny
student record book number 276890, 305765

supervisor
Zinelabidine Leghelimi

WARSAW April 2, 2025

Contents

1. Introduction

The primary objective of this project was to compare the performance of several convolutional neural network (CNN) architectures on the image classification task using the CINIC-10 dataset. Beyond simple benchmarking, we focused on understanding how key training components (hyperparameters, data augmentations, few-shot learning, and model ensembling) impact overall model performance in practice.

We used these three models:

- ResNet (ResNet-50),
- EfficientNetV2,
- ConvNeXt.

All models were initialized with pretrained ImageNet weights and fine-tuned using the TensorFlow framework within the Google Colab environment.

2. Theoretical Background and Literature Review

This project is based on the most popular and modern ideas in deep learning. Before starting the experiments, we studied several important research papers that helped us choose the models and training methods.

2.1. Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a special kind of computer program that can look at pictures and find patterns in them. For example, it can learn to tell the difference between a cat and a dog by looking at many photos. CNNs are now widely used in many tasks like face recognition, object detection, and medical image analysis.

2.2. Models

In our project, we tested three popular CNN models:

- ResNet-50[1]: This is an older but still strong model. It has a smart design that helps it learn even when it is very big. It's often used by researchers and engineers.
- EfficientNetV2[2]: This model is newer. It is faster and smaller than ResNet, but still very good. It learns quickly and works well on many tasks.
- ConvNeXt[3]: This is one of the newest models. It uses ideas from both CNNs and transformers (a different kind of model that's good with language and images). It tries to combine the best of both worlds.

2.3. Ensembling

Ensembling means using several models together instead of just one. Each model gives its answer, and we combine them to make a final decision. This usually gives better

results. There are different methods to combine the results of different models. Methods used in our project:

- **Hard Voting:** Each model votes for a class and the majority wins.
- **Soft Voting:** We take the average of the probability scores from all models and choose the most likely class.
- **Stacking:** We train another model that learns how to best combine the predictions of the models.

2.4. Data Augmentation

This means we take the training images and make small changes, like rotating them or making them brighter. This helps the model learn better because it sees more variety.

2.5. Few-Shot Learning

Sometimes we don't have many images to train with. We tested how well our models work when we only give them 10 or 50 pictures per category. Thanks to using pretrained model (ImageNet - that already learned from millions of pictures), they still did a pretty good job.

2.6. Literature for theoretical background

The following papers and articles served as the theoretical background:

1. ResNet: Deep Residual Learning for Image Recognition
2. EfficientNetV2: EfficientNetV2: Smaller Models and Faster Training
3. ConvNeXt: A ConvNet for the 2020s
4. Ensembling: Efficient Adaptive Ensembling of Vision Transformers
5. Ensembling: Ensembling: Voting and Stacking

These resources helped us with our model choices, architecture understanding, and ensembling strategies.

3. Methodology and Experimentation

In this chapter, we explain how we trained our models and what steps we followed. We tested different settings to see what works best. This helped us learn how different choices can affect the results.

All experiments in this project were done using the CINIC-10 dataset, which is a collection of color images from 10 different classes (like airplane, car, dog, cat, etc.). It is a good test for models that classify everyday objects.

Our deep CNN model is based on transfer learning, where we use a pre-trained base model and fine-tune the classification layers. The architecture follows these steps:

- The base model is pre-trained on a large dataset and is set to non-trainable (`trainable=False`) to retain its learned features. Weight of our models are taken from optimization of ImageNet.
- Global Average Pooling layer reduces the spatial dimensions while retaining essential features.
- Dropout layer with a specified dropout rate is added to prevent overfitting. We are analyzing dropout rate further in the report.
- A fully connected layer with 512 neurons, ReLU activation[1], and L1 regularization. We are analyzing L1 regularization parameter further in the report as well.
- The final output layer is a softmax layer with a number of units equal to the number of classes, producing class probabilities.

The model is optimized using the Adam optimizer[4], which is an adaptive gradient-based optimization algorithm. We trained each model in each experiment for 10 rounds (called epochs) and checked the accuracy.

3.1. Hyperparameters

Hyperparameters are settings we choose before training the model. They control how the model learns. We tested different values for each one to find which works best.

Here are the hyperparameters we tested:

- Learning rate: This controls how fast the model learns. If it's too high, the model can get confused. If it's too low, the model learns too slowly.
- Batch size: This tells the model how many images to look at at once during training.
- Dropout: This is a trick to help the model not memorize the training data too much. It ignores some neurons in the model randomly during training.
- L1 Regularization: This is another way to stop the model from overfitting (learning the training data too perfectly and doing badly on new data).

For each test, we changed only one hyperparameter at a time and kept the others the same.

3.2. Data Augmentation

Data augmentation means changing the images in small ways so the model sees more variety. This helps it learn better.

We tested these data augmentations:

- Rotation: turning the image a little.
- Brightness: making the image lighter or darker.
- Zoom: zooming in on the image.
- CutMix: mixing two images together

3.3. Few-Shot Learning

Sometimes we don't have many pictures for training. We tested how well our models can learn when given only a few images per class. In our case that was 10, 50 and 200 images per class.

3.4. Ensembling

Instead of using only one model, we also tried combining models. This is called ensembling.

We used three methods:

- Hard voting: each model gives its answer, and the one with the most votes wins.
- Soft voting: we average the confidence of each model for each class.
- Stacking: we train another model (a simple one called logistic regression) to learn how to combine the others.

4. Results and Discussion

This section presents an analysis of the results obtained from the conducted experiments. Each hyperparameter was individually varied while maintaining a default configuration as a reference point:

- learning rate = 0.005,
- batch size = 128,
- dropout = 0.5,
- L1 regularization = 0.0,
- data augmentation = none,
- no few-shot learning.

Each configuration was trained for 10 epochs. The relatively low number of epochs, while sufficient for observing general trends, likely limited the model's ability to fully converge, especially for configurations involving high regularization or low learning rates.

4.1. Hyperparameters

4.1.1. Learning Rate

Learning rate was tested with four values: 0.0001, 0.0003, 0.0005, and 0.001. Higher learning rates resulted in better training accuracy, but validation accuracy became unstable for 0.001 rate. This aligns with the common observation that too large a learning rate leads to overshooting and poor generalization. This observation aligns with findings by Bengio [5].

The comparison plot across models [4.1][4.2][4.3][4.4] indicates that learning rates of 0.003 and 0.005 generally yielded the best performance[4.5]. These values offered a good

balance between learning speed and generalization, especially under the constraint of only 10 training epochs, which limits the benefit of slower learning. This result is consistent with best practices in CNN fine-tuning [6].

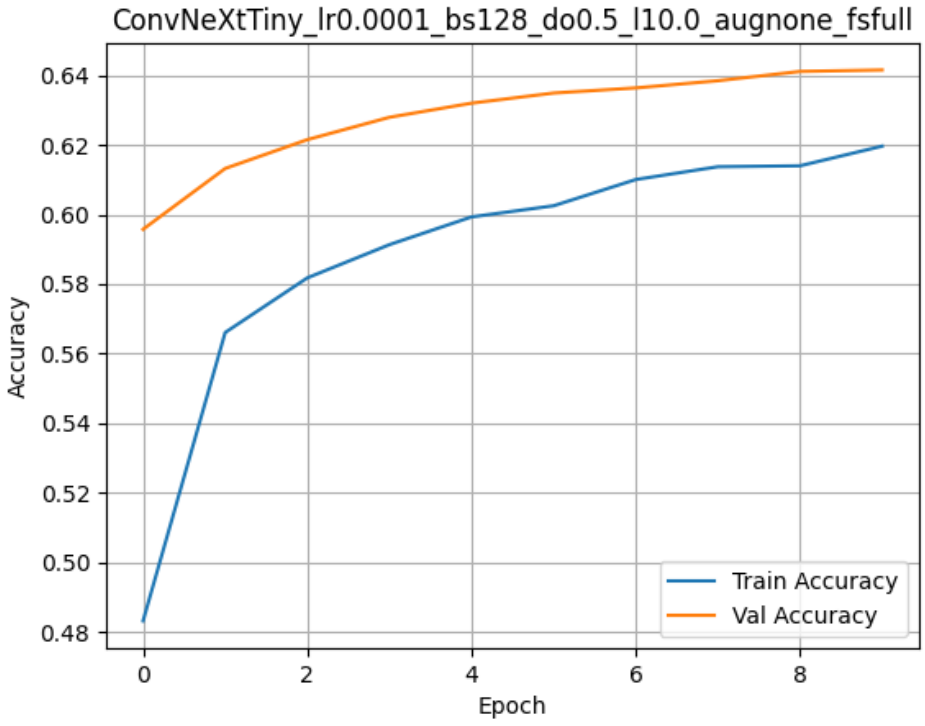

Figure 4.1. Chart for Learning Rate = 0.0001

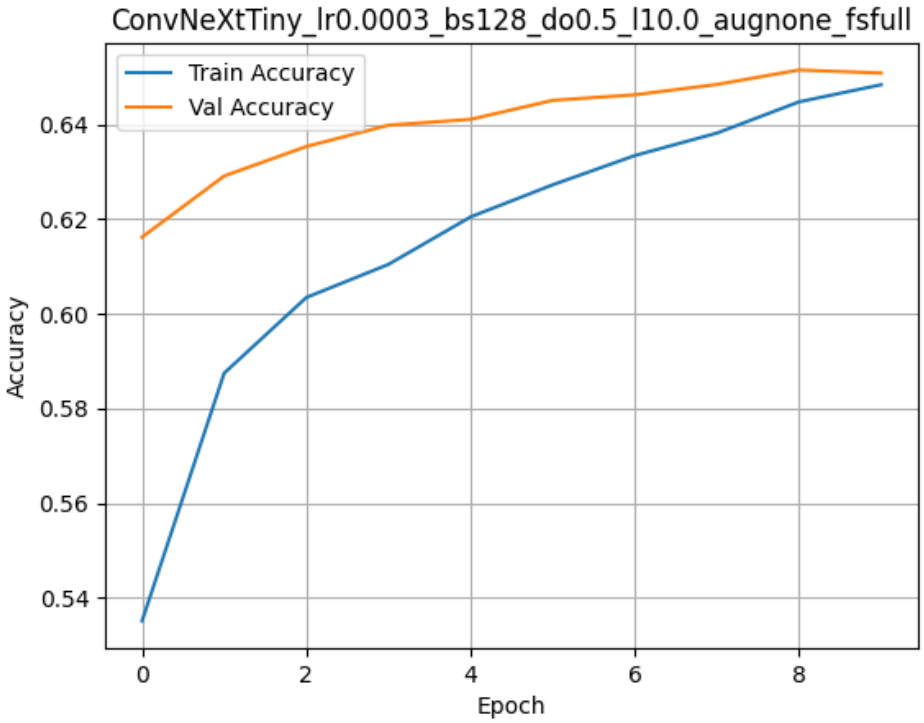

Figure 4.2. Chart for Learning Rate = 0.0003

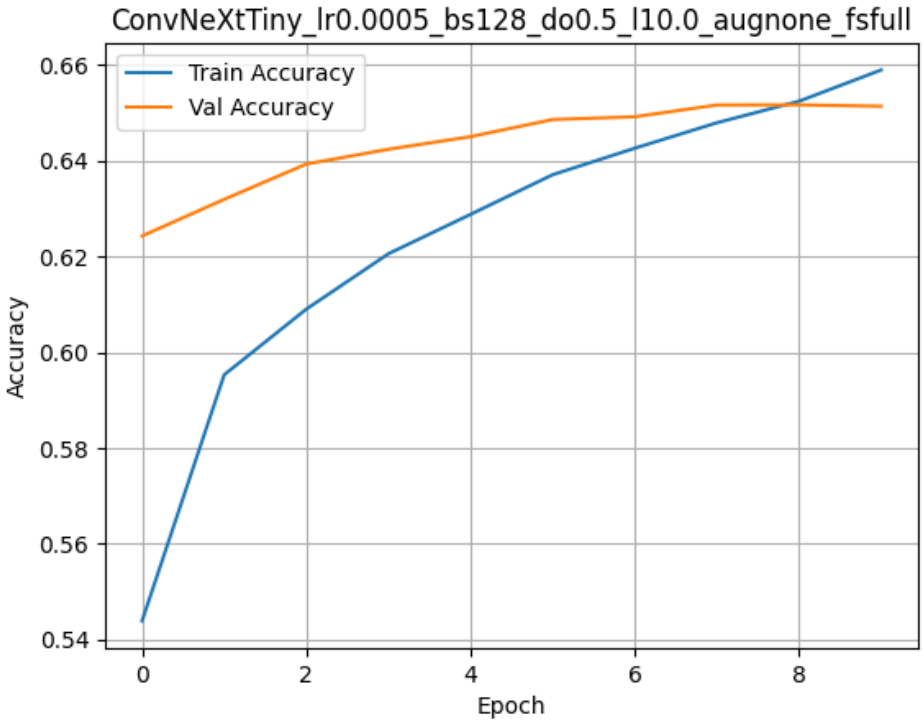

Figure 4.3. Chart for Learning Rate = 0.0005

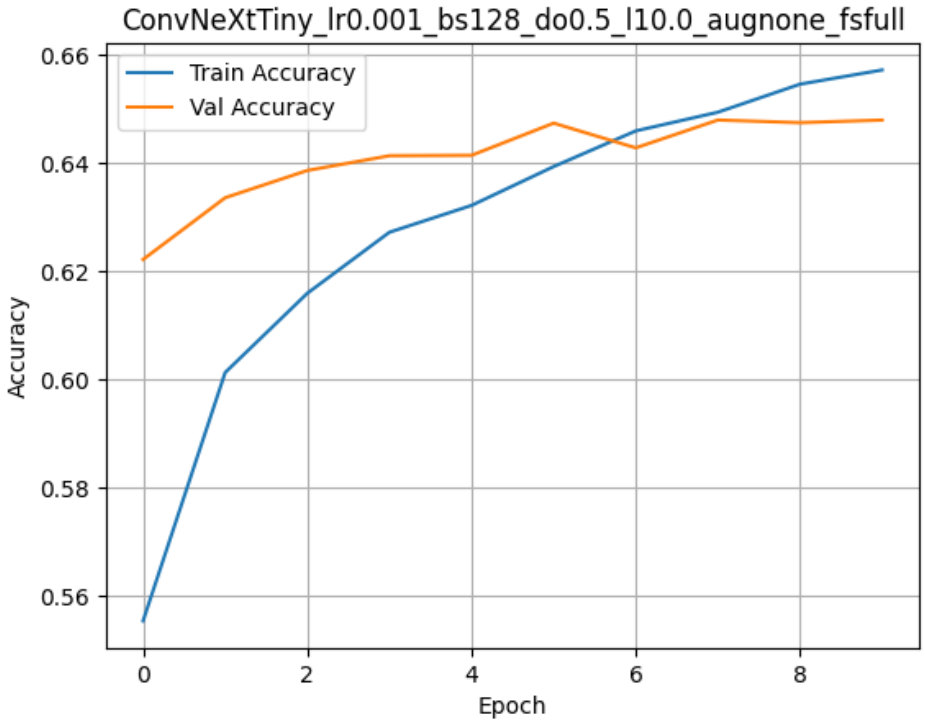

Figure 4.4. Chart for Learning Rate = 0.001

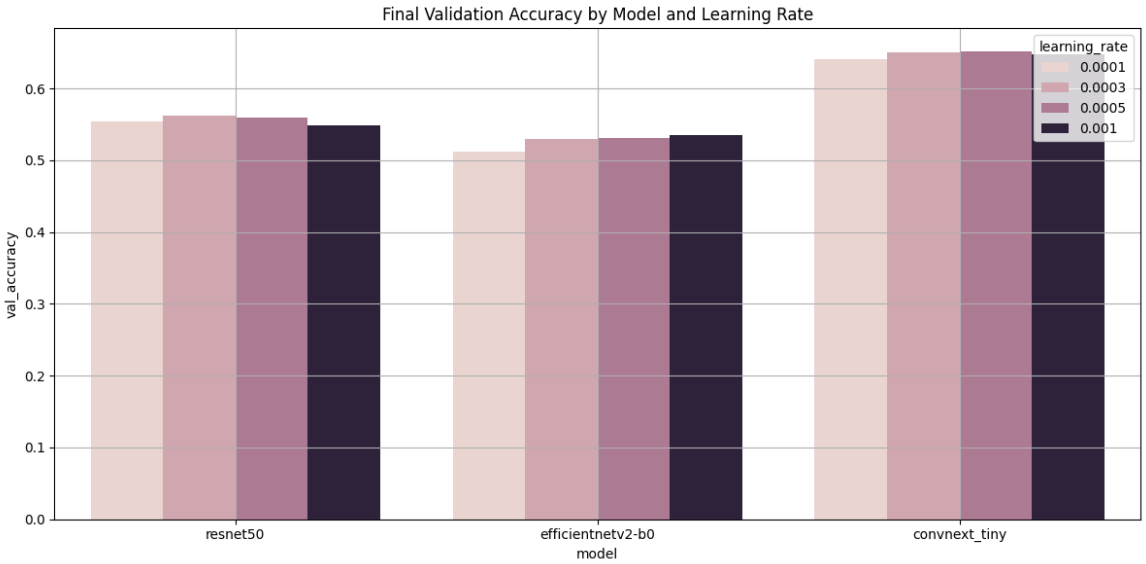

Figure 4.5. Comparison Chart for Learning Rates

4.1.2. Batch Size

Batch sizes of 32, 64, and 128 were compared. As expected, the training and validation performance was relatively insensitive to batch size, which is consistent with findings from Keskar et al. [7].

The limited number of epochs probably contributed to the result of the experiments, as batch size can have more pronounced effects in longer training regimes. The plots suggest, that there is still room for better convergence[4.6][4.7][4.8]. In our setup, it mostly affected memory usage and iteration speed rather than final accuracy. The best results were for batch size equal to 128[4.9].

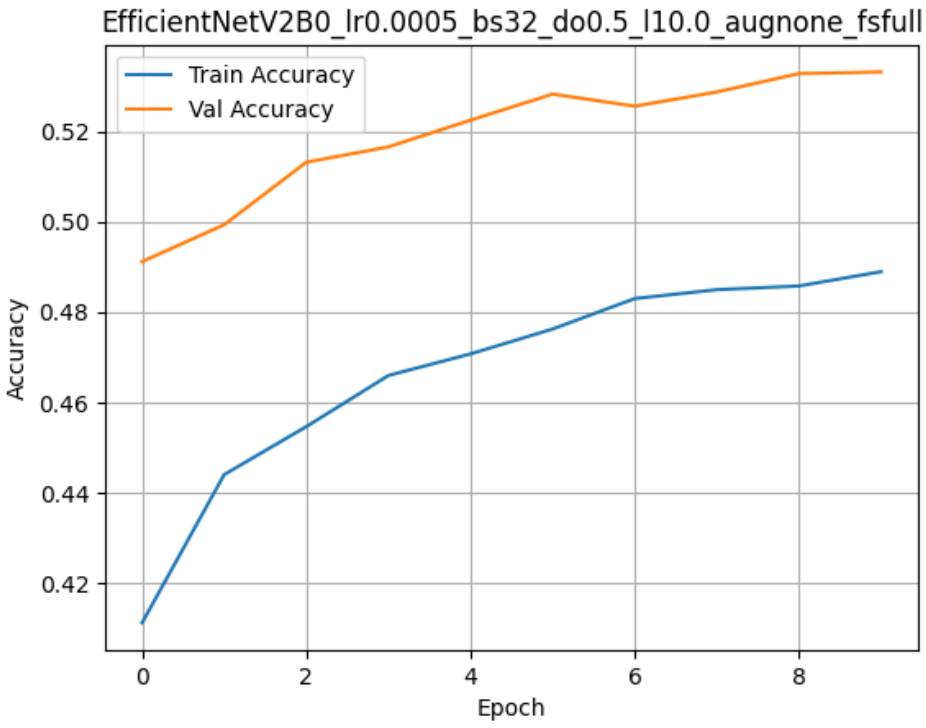

Figure 4.6. Chart for Batch Size = 32

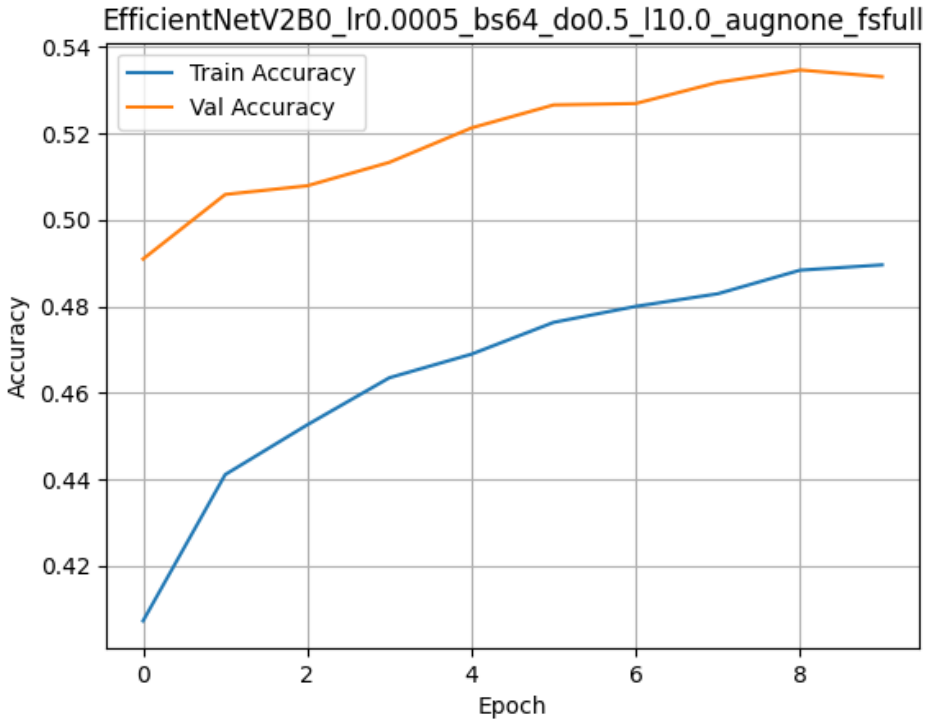

Figure 4.7. Chart for Batch Size = 64

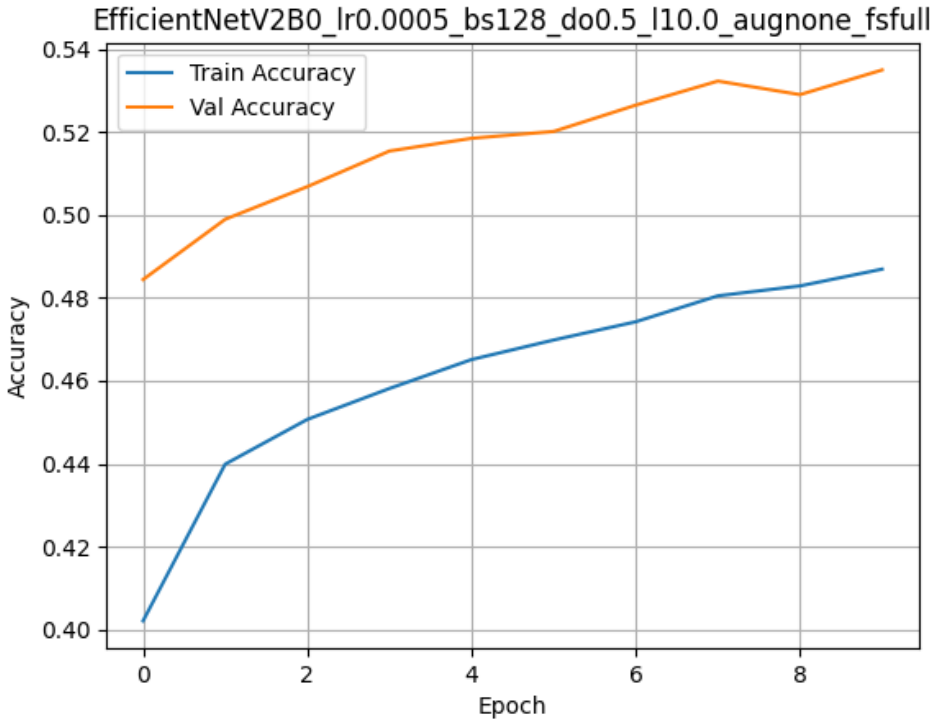

Figure 4.8. Chart for Batch Size = 128

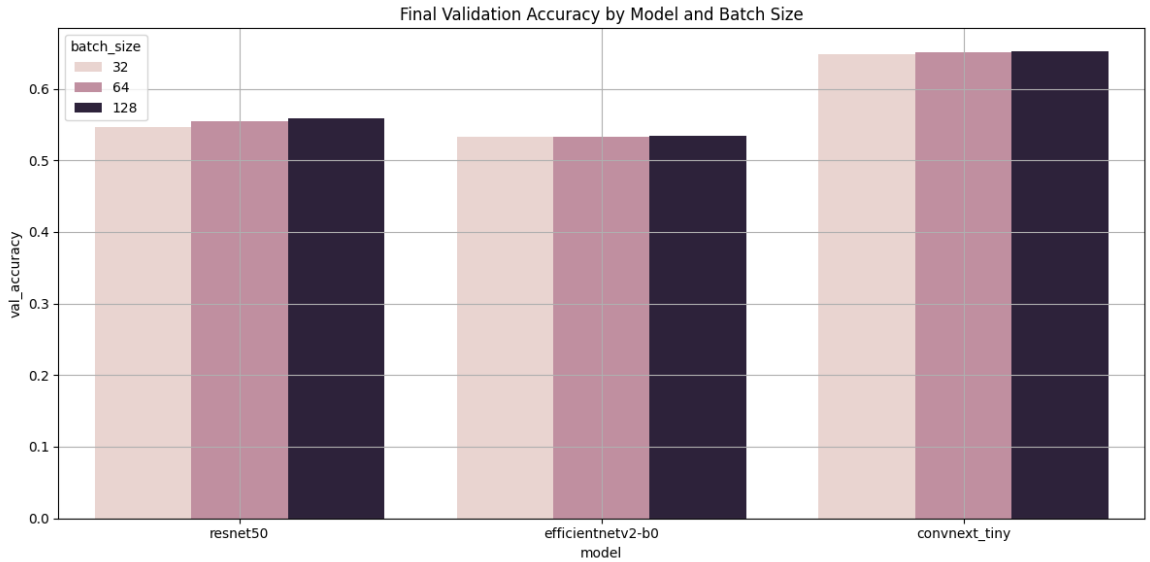

Figure 4.9. Comparison Chart for Batch Sizes

4.1.3. Dropout

Dropout rates of 0.3, 0.5, and 0.7 were tested. The results show that a dropout of 0.3 consistently outperformed 0.5 and 0.7. A high dropout rate (0.7) caused clear underfitting[4.12], with reduced training and validation performance, while 0.5 slightly underperformed compared to 0.3[4.10][4.11].

The final comparison [4.13] across all models confirms that lower dropout values led to better performance, suggesting that lighter regularization was more appropriate. Given the short training time, aggressive dropout was unnecessary and even made results worse. This is why our result is a bit different from what Srivastava et al. [8] suggested in their dropout paper.

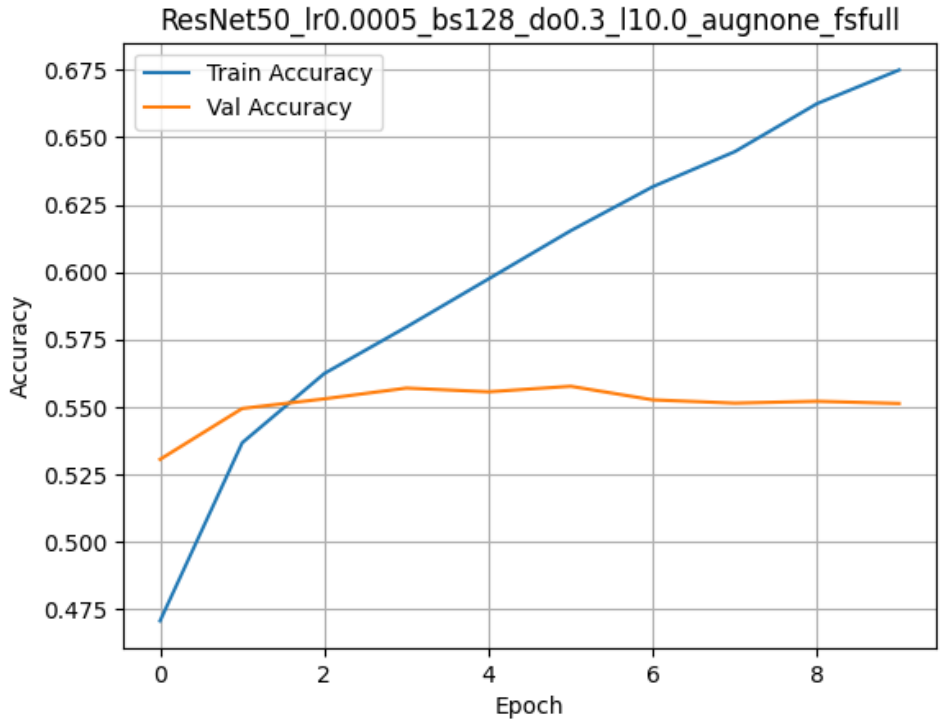

Figure 4.10. Chart for Dropout = 0.3

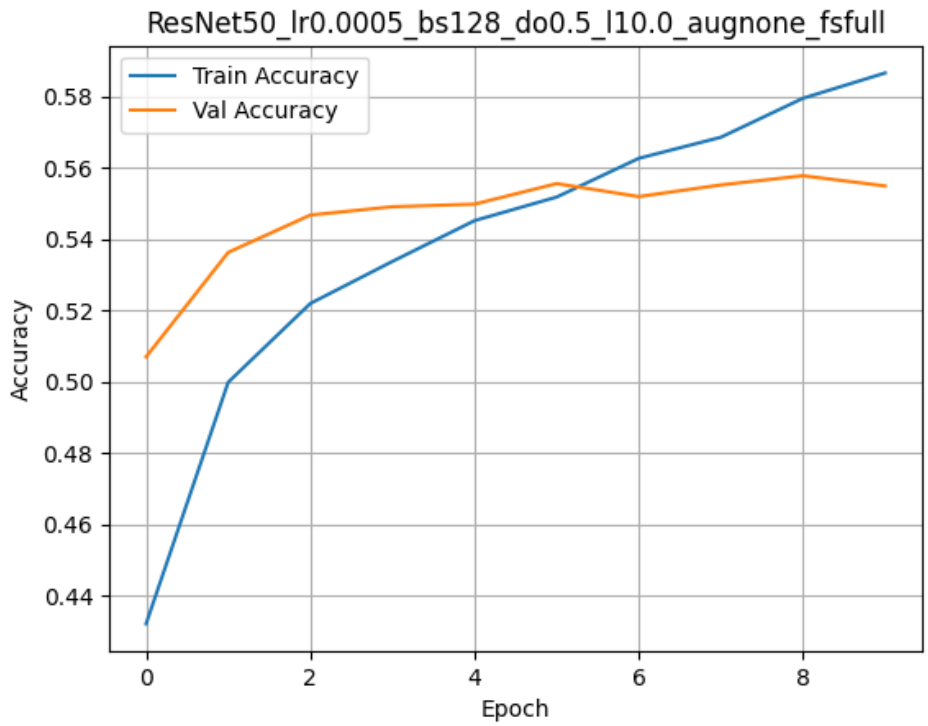

Figure 4.11. Chart for Dropout = 0.5

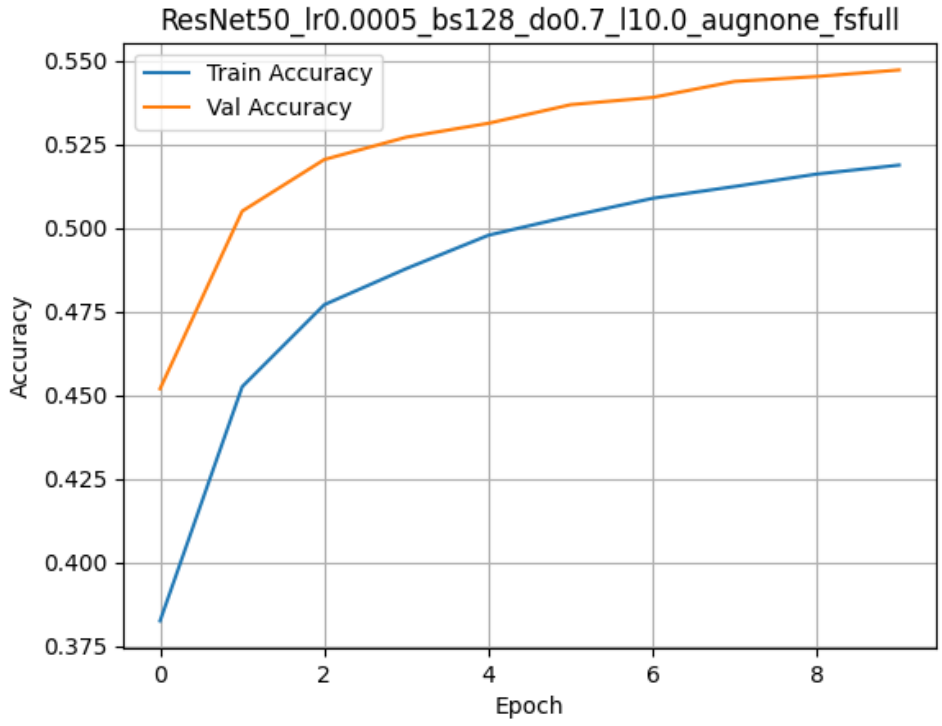

Figure 4.12. Chart for Dropout = 0.7

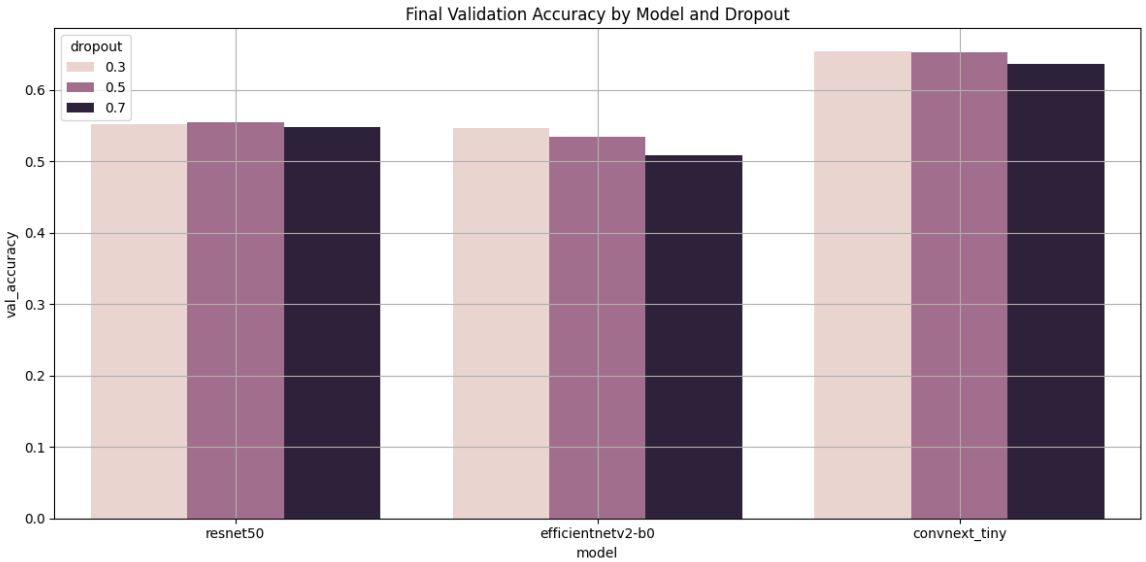

Figure 4.13. Comparison Chart for Dropout Rates

4.1.4. L1 Regularization

L1 regularization was tested at values of 0.0, 0.0001, and 0.0005. Results indicate that increasing L1 penalty led to decreased accuracy, most notably with 0.0005[4.17].

This means that with short training (10 epochs), L1 regularization can limit the model too much and stop it from learning the training data well. It may only be helpful when training is longer or when the model is overfitting a lot[4.14][4.15][4.16].

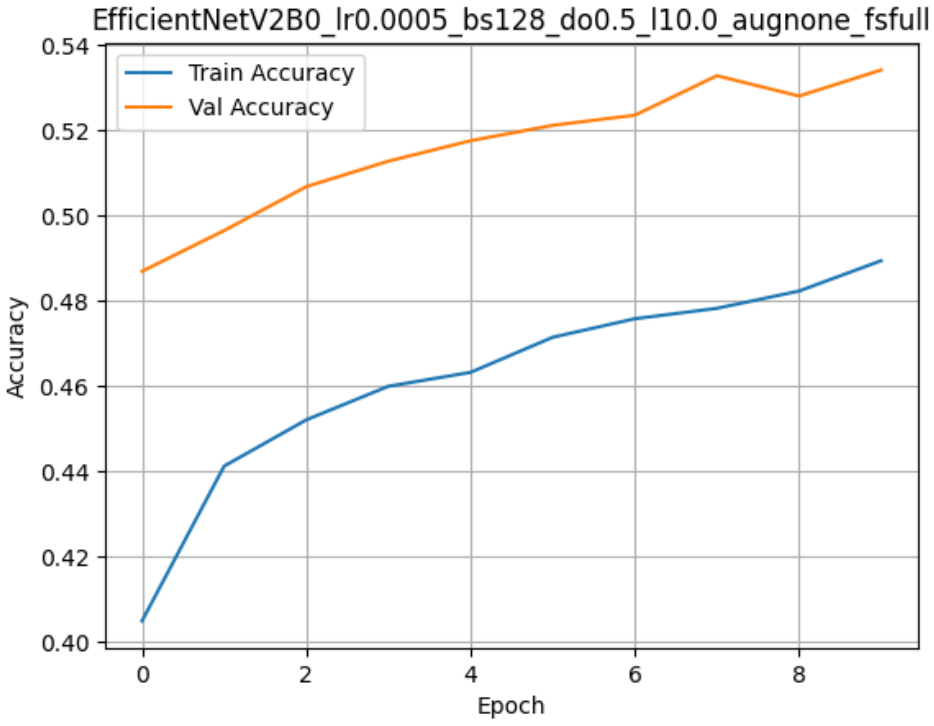

Figure 4.14. Chart for L1 Regularization = 0.0

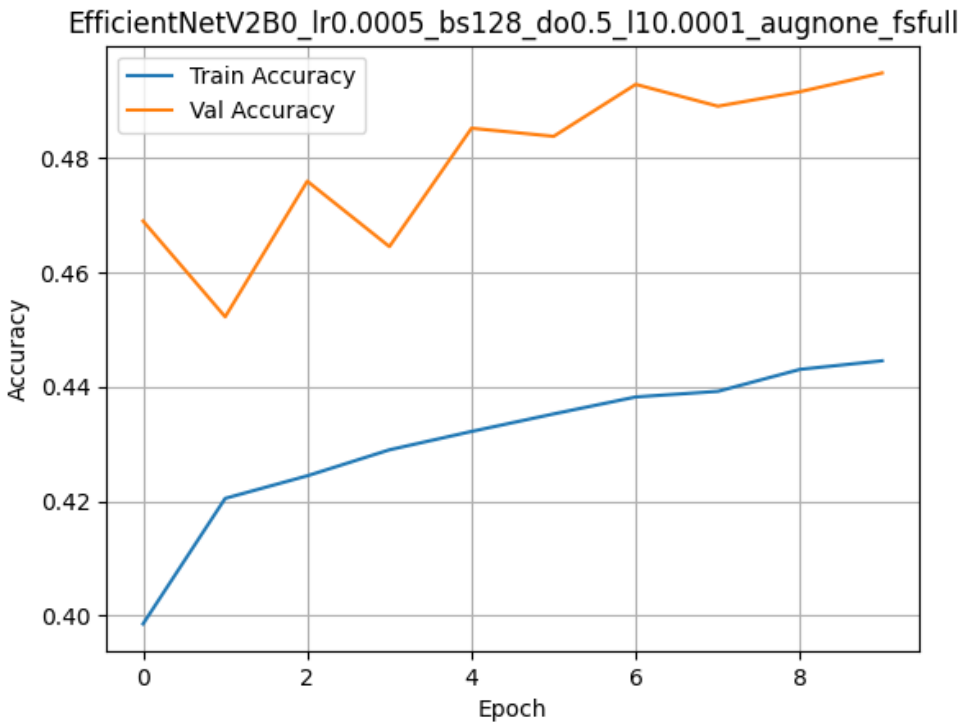

Figure 4.15. Chart for L1 Regularization = 0.0001

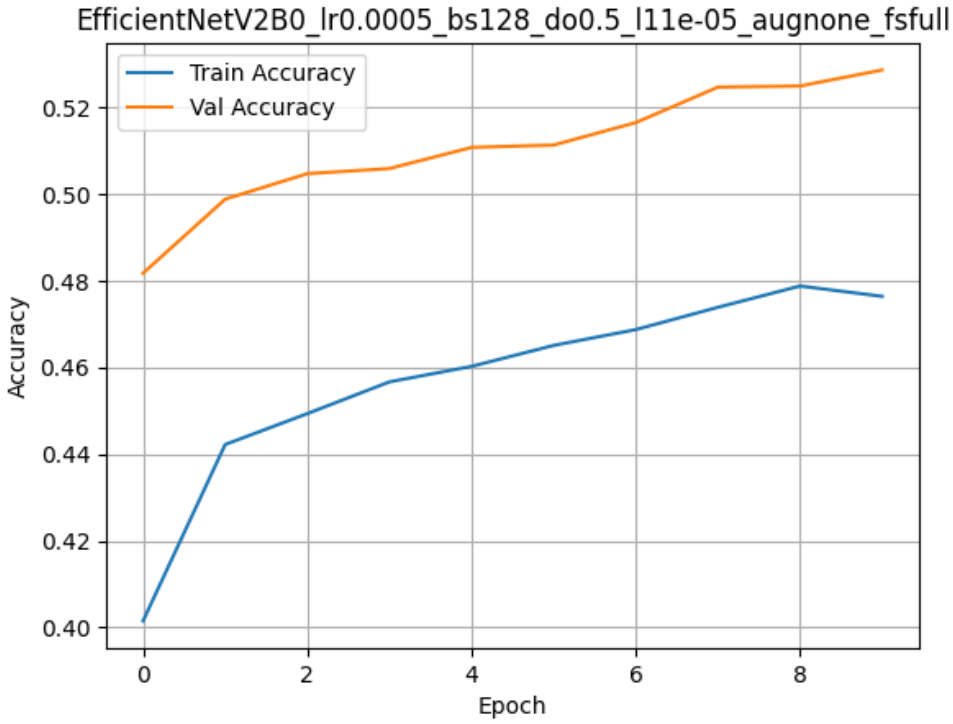

Figure 4.16. Chart for L1 Regularization = 0.0005

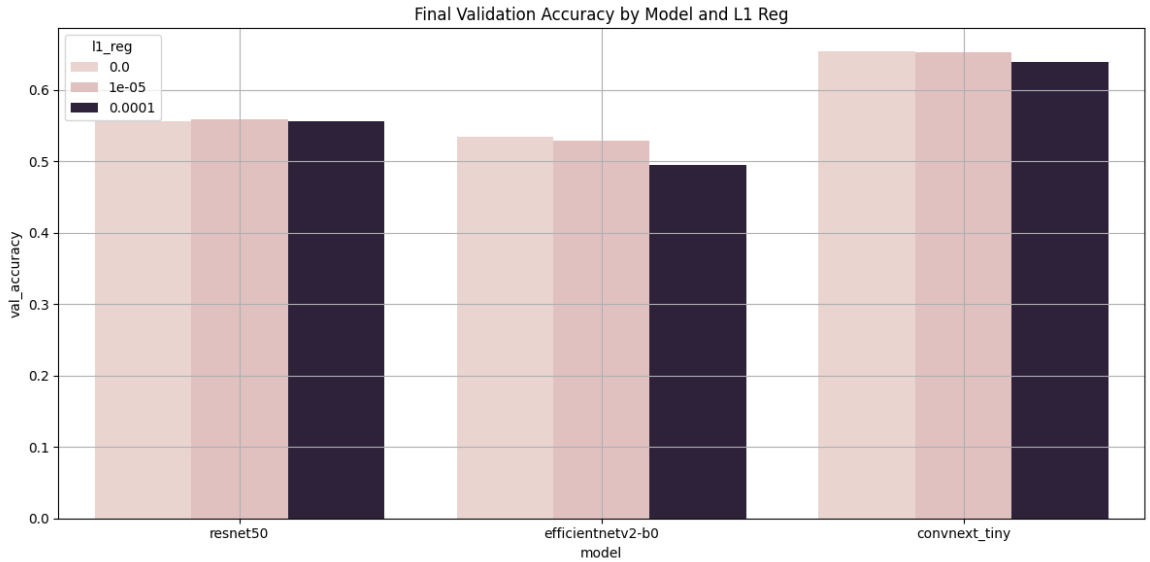

Figure 4.17. Comparison Chart for L1 Regularization

4.2. Augmentation

Techniques like Zoom and Contrast offered modest improvements, while CutMix underperformed in this setting, possibly due to the short training regime and the additional complexity it introduces.

This suggests that for limited training budgets, simpler augmentations like rotation may be more reliable [9], [10]. Furthermore, CutMix has shown strong results in other studies [11], but typically in conjunction with longer training or additional regularization strategies.

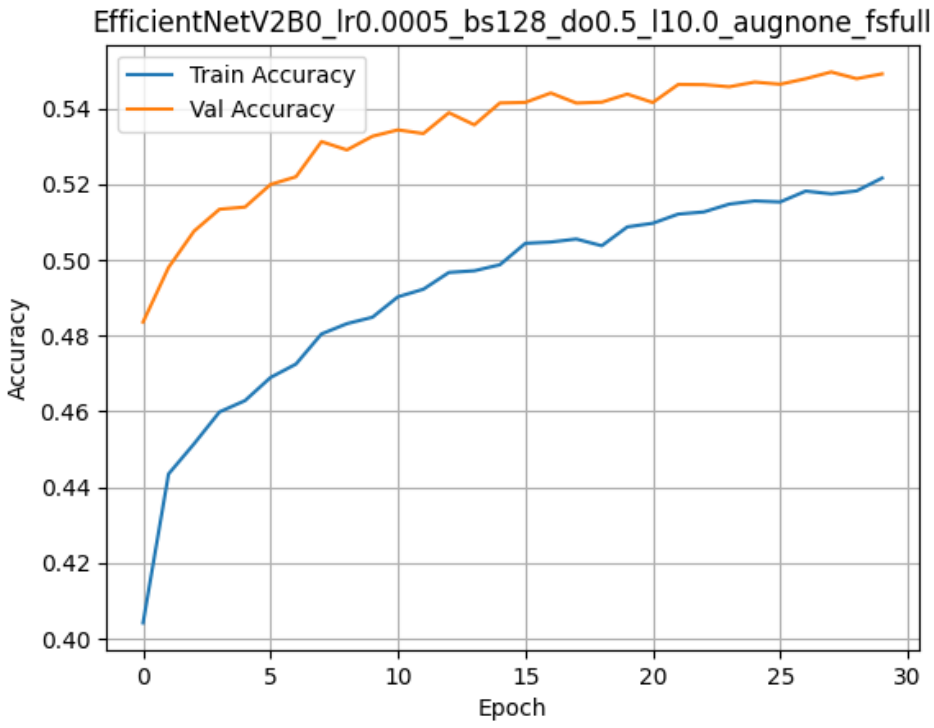

Figure 4.18. Chart for Augmentation = None

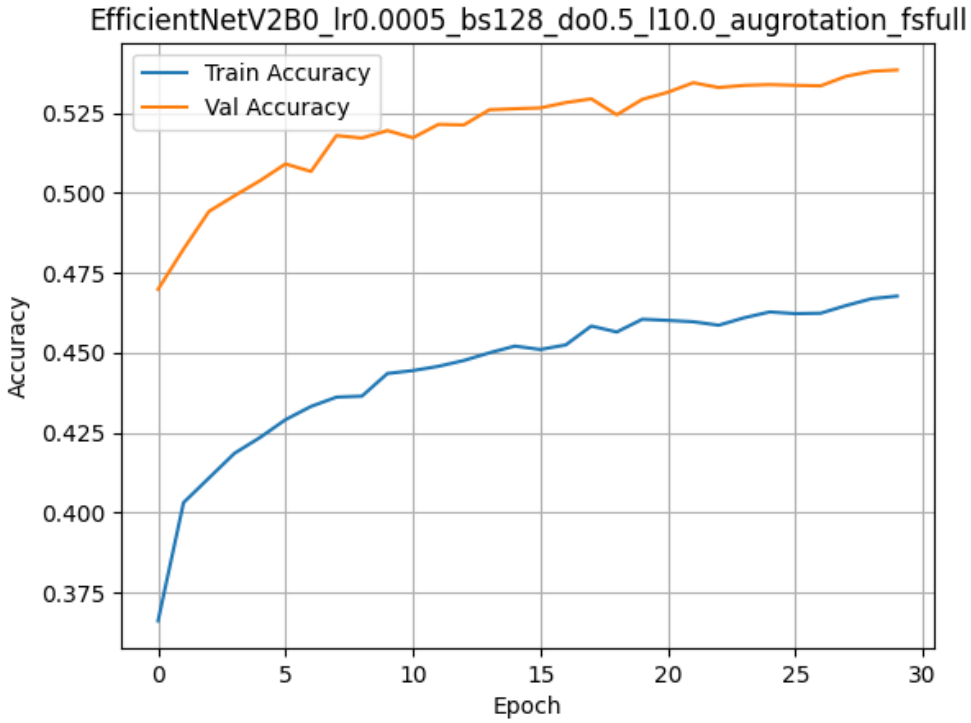

Figure 4.19. Chart for Augmentation = Rotation

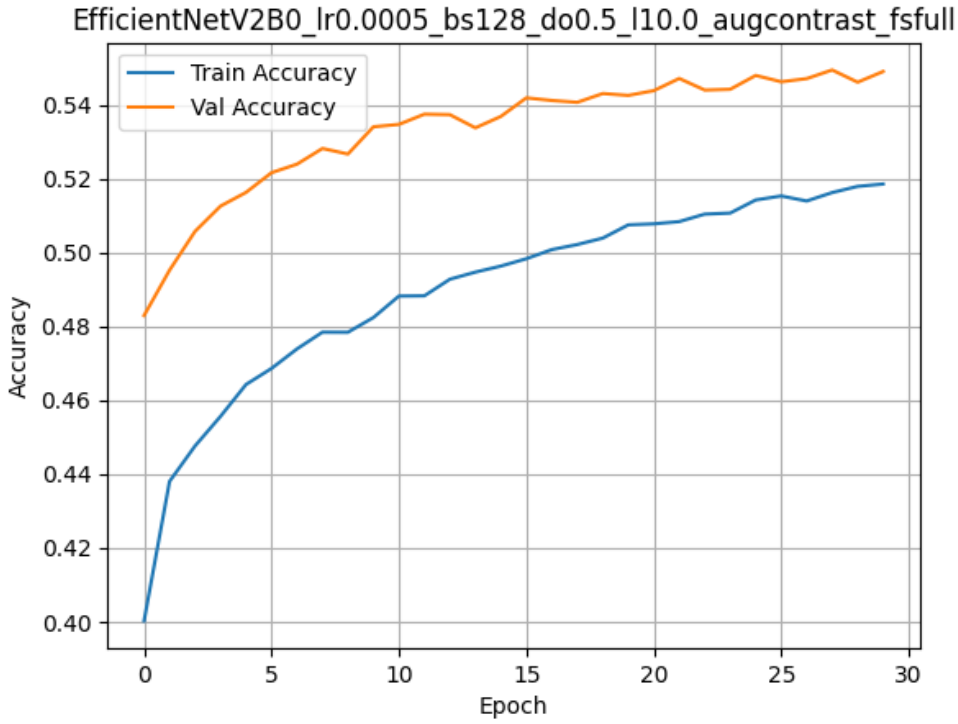

Figure 4.20. Chart for Augmentation = Contrast

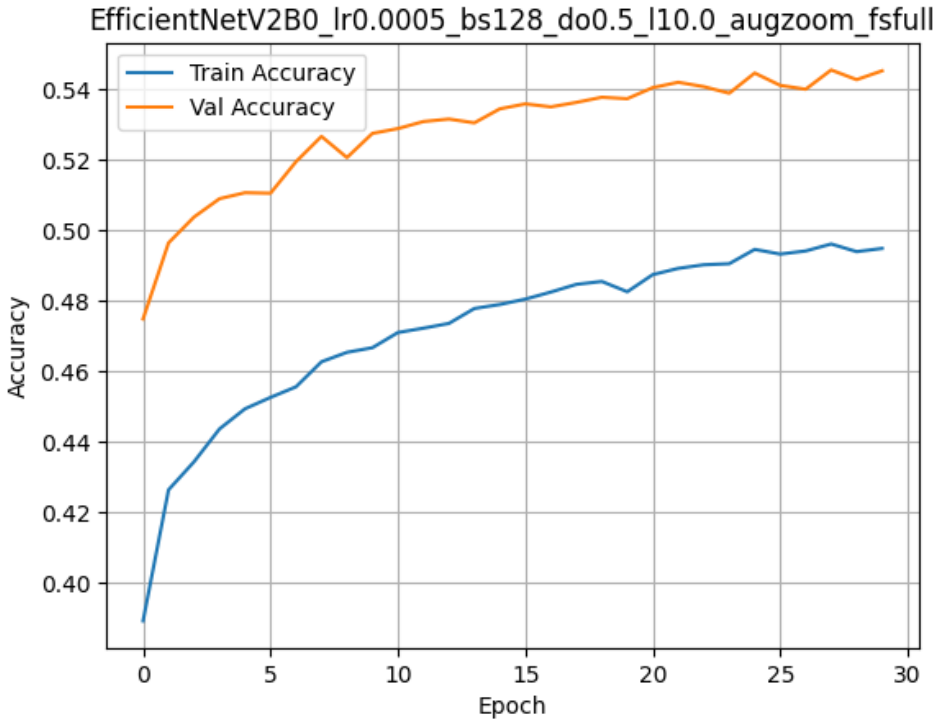

Figure 4.21. Chart for Augmentation = Zoom

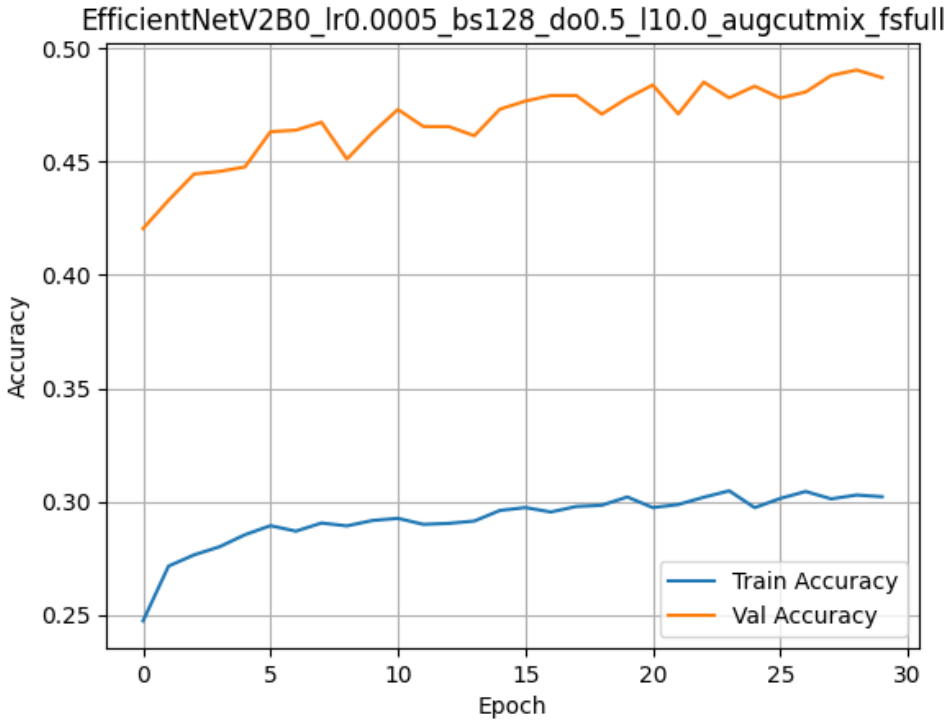

Figure 4.22. Chart for Augmentation = CutMix

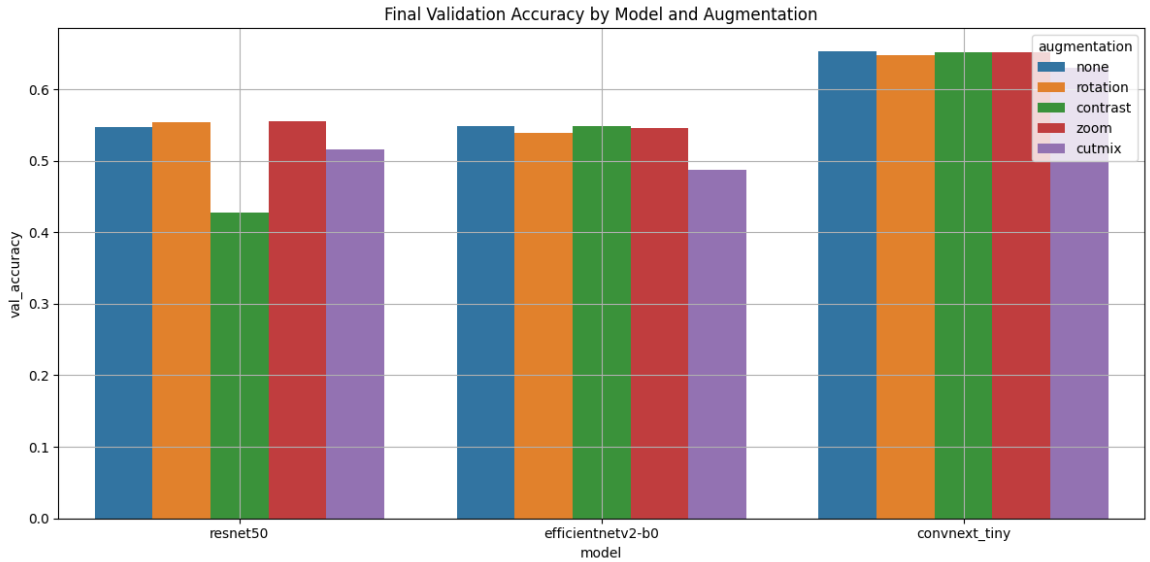

Figure 4.23. Comparison Chart for Augmentation Methods

4.3. Few Shot Learning

Models were trained with 10, 50, and 200 images per class. As expected, accuracy improved with more data. The ConvNeXt plots show rapid gains from 10 to 50 images[4.24][4.25], and further improvement with 200 samples[**??**].

Final comparison confirms that all architectures benefited from additional data, but pretraining (weights from ImageNet) enabled decent performance even in extremely low-data settings (10 images per class)[**??**]. This highlights the effectiveness of transfer learning for few-shot scenarios, as supported by Yosinski et al. [12].

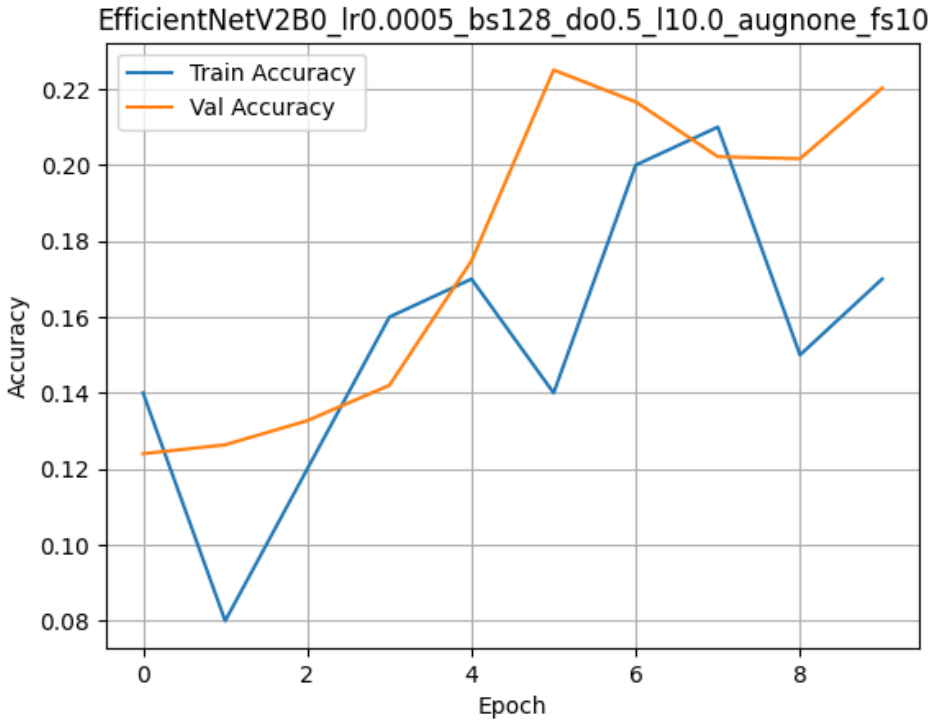

Figure 4.24. Chart for Few-Shot = 10 Images per Class

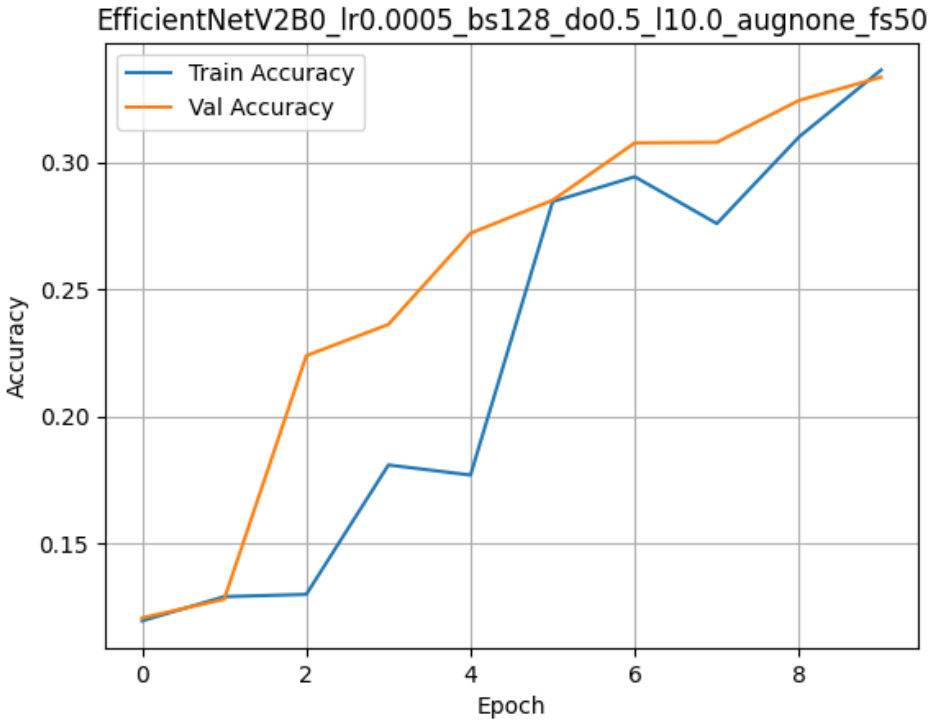

Figure 4.25. Chart for Few-Shot = 50 Images per Class

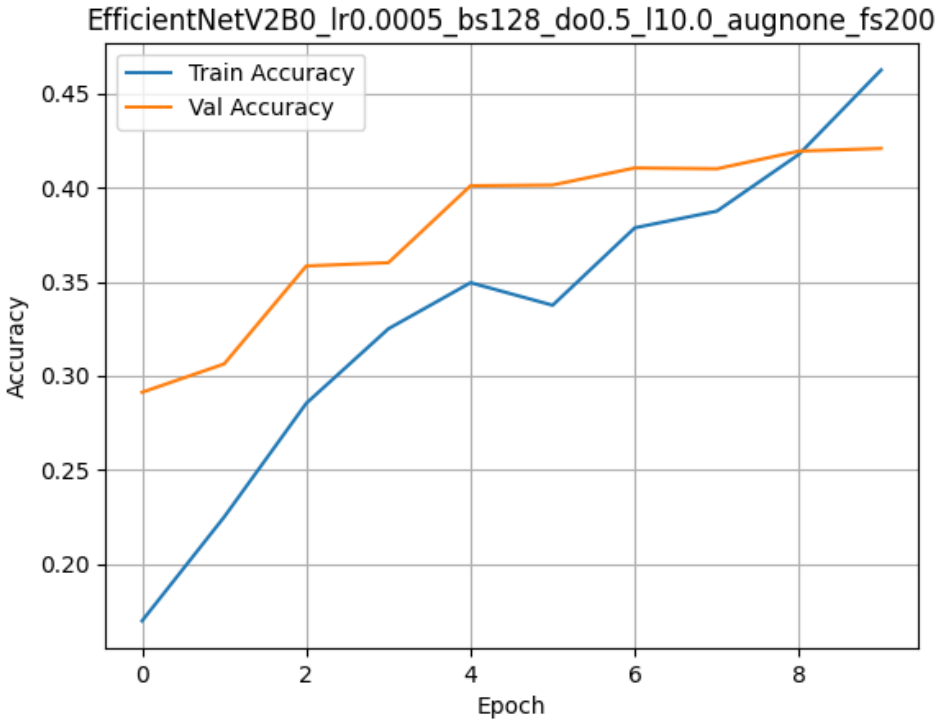

Figure 4.26. Chart for Few-Shot = 200 Images per Class

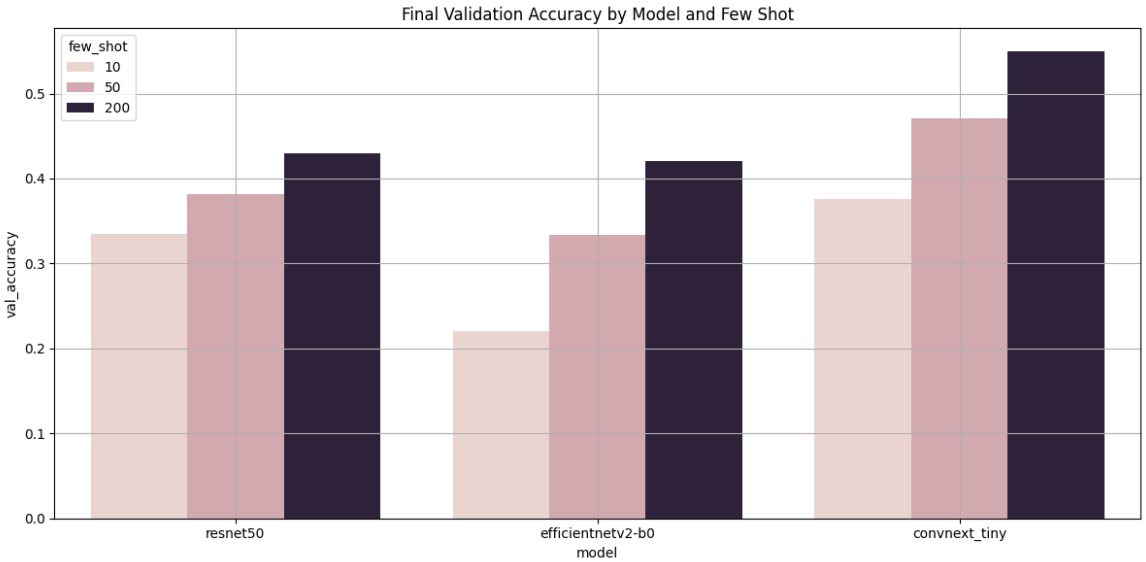

Figure 4.27. Comparison Chart for Few-Shot Learning

4.4. Ensembling

We implemented 3 different ensembling methods. We used hard-voting, soft-voting and stacking. For stacking method we used Logistic Regression. Firstly, we selected 3 best deep-CNN models, that we trained earlier. One for each depp-CNN. Then we checked how the methods perform. The best results were obtained with stacking method and soft-voting, although hard-voting also got good results[4.28]. All results were better than without ensembling[4.1][5.1]

Method	Test Accuracy
Soft Voting	0.6730
Hard Voting	0.6323
Stacking	0.6785

Table 4.1. Validation accuracy for different ensemble methods

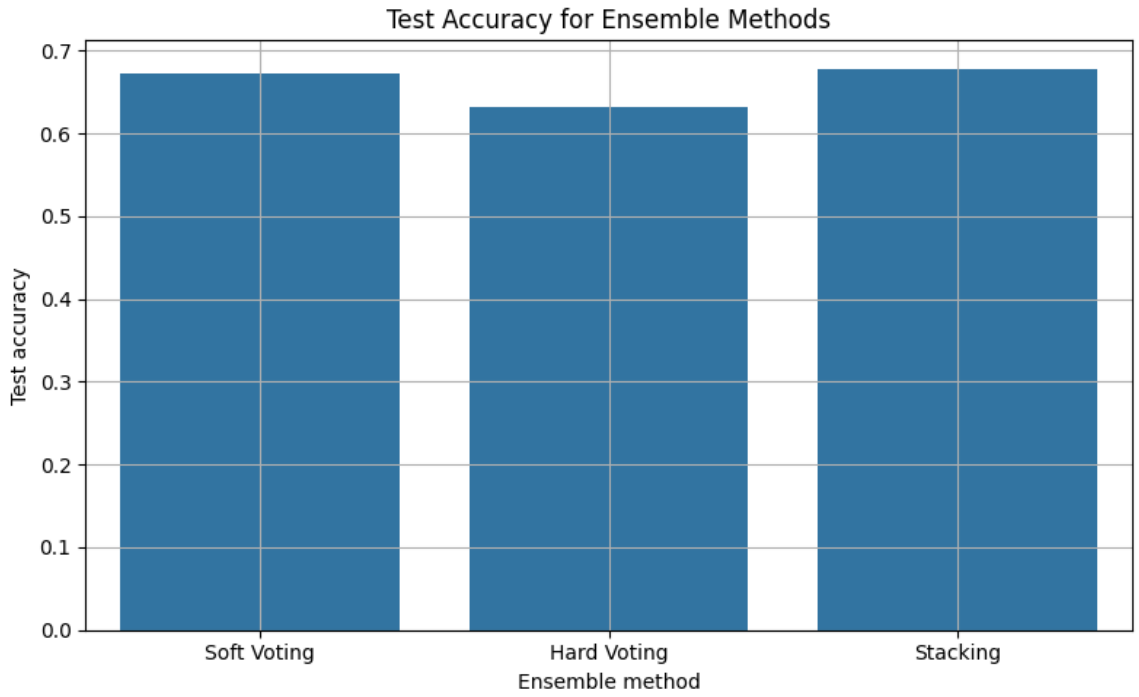

Figure 4.28. Comparison Chart for Ensemble Methods

5. Conclusions and Future Work

In this project, we tested three powerful models for image classification: ResNet-50, EfficientNetV2, and ConvNeXt. We used the CINIC-10 dataset and tried different training settings to see what works best.

Model	LR	BS	Dropout	L1	Aug.	Few-Shot	Test Acc
EfficientNetV2-B0	0.0005	128	0.5	0.0	None	Full	0.5540
ResNet50	0.0003	128	0.5	0.0	None	Full	0.5645
ConvNeXt-Tiny	0.0005	128	0.5	0.0	Contrast	Full	0.6587

Table 5.1. Best results for every model

5.1. Conclusions

- ConvNeXt got the best results for test accuracy[5.1].
- EfficientNetV2 gave better results than ResNet-50 in most experiments.
- Some simple changes like lowering dropout or using rotation as augmentation helped a lot.
- Few-shot learning worked well, even with just 10 images per class, thanks to pre-trained models.
- Ensembling (combining models) improved performance. The best results came from the stacking method, which mixes models in a smart way.

5.2. Future Work

- Try transformer-based models like Vision Transformers (ViT).
- Use more specific datasets, for example, medical or industrial images, to see how well the models transfer.
- Train for more epochs. Methods like CutMix or higher regularization work better with longer training.
- Calculate our algorithms multiple times in order to get mean accuracy of models and check variation of our results. It could be preferable to choose model with smaller accuracy, but smaller variance of the results.
- Try ensembling for more deep-CNN models.

6. Application Instruction

1. Open the `.ipynb` file from the ZIP archive in Google Colab
   First, unzip the project folder on your computer. Then open Google Colab, click `File` $\rightarrow$ `Upload notebook`, and select the $\texttt{cnn}_r zepinski_w olny.ipynb file from the extracted folder.$
   $Make sure you select a GPU runtime.$
   ($Click$ `Runtime` $\rightarrow$ `Change runtime type` $\rightarrow choose$ `GPU`).
2.
3. Download JSON with Kaggle API token
   Go to kaggle.com/account, scroll to `API`, and click `Create New API Token`.

This will download a file called `kaggle.json` — you will need it to access datasets from Kaggle.

4. Run the notebook

   Run each cell step by step (from top to bottom).

   When the first cell asks you to upload a file, upload the `kaggle.json` file with your Kaggle API token.

File `ensemble.ipynb` is not run on Google colab. It can be run on your own computer. It is required to download have data downloaded in proper folder `./content/cinic10` and define paths to keras models, that should be used in ensembling.

References

[1] A. F. Agarap, "Deep learning using rectified linear units (relu)", *CoRR*, vol. abs/1803.08375, 2018. arXiv: 1803.08375. [Online]. Available: http://arxiv.org/abs/1803.08375.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[3] M. Tan and Q. V. Le, "Efficientnetv2: Smaller models and faster training", *CoRR*, vol. abs/2104.00298, 2021. arXiv: 2104.00298. [Online]. Available: https://arxiv.org/abs/2104.00298.

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980.

[5] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures", *Neural Networks: Tricks of the Trade*, 2012, https://arxiv.org/abs/1206.5533.

[6] L. N. Smith, "Cyclical learning rates for training neural networks", *WACV*, 2017, https://arxiv.org/abs/1506.01186.

[7] N. S. Keskar *et al.*, "On large-batch training for deep learning: Generalization gap and sharp minima", 2016, https://arxiv.org/abs/1609.04836.

[8] N. Srivastava *et al.*, "Dropout: A simple way to prevent neural networks from overfitting", *Journal of Machine Learning Research*, 2014, https://jmlr.org/papers/v15/srivastava14a.html.

[9] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning", *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019. [Online]. Available: https://arxiv.org/abs/1901.11196.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017, Originally published in NeurIPS 2012. [Online]. Available: https://papers.nips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

[11] S. Yun *et al.*, "Cutmix: Regularization strategy to train strong classifiers with localizable features", in *ICCV*, https://arxiv.org/abs/1905.04899, 2019.

[12] J. Yosinski *et al.*, "How transferable are features in deep neural networks?", *NeurIPS*, 2014, https://arxiv.org/abs/1411.1792.