# CNN Architectures for Image Classification: From VGG to ConvNeXt

**Clément Corbeau–Izorche**
clci@kth.se

**Antonin Roy**
antoninr@kth.se

**Sneha Agrawal**
snehaa@kth.se

**János Bence, Monori**
monori@kth.se

## Abstract

This project explores how to train convolutional neural networks (CNNs) from scratch for image classification, using standard benchmarks such as CIFAR-10, CIFAR-100, and ImageNette. We begin with a simple VGG-style model and progressively improve performance through architectural upgrades and regularization techniques, including dropout, weight decay, and data augmentation. We then extend the baseline to ResNet-20 with residual connections, and integrate attention mechanisms like Squeeze-and-Excitation (SE) and Convolutional Block Attention Module (CBAM). ConvNeXt-Tiny, a transformer-inspired CNN, is also evaluated under low-resolution conditions. It achieves a test accuracy of 86.42% on ImageNette 160px, showing the benefits of its inspiration from transformers. Finally, we attempt to address label noise by replacing standard cross-entropy with Symmetric Cross-Entropy, which aims to improve training stability and robustness in noisy label settings. The best-performing VGG-based model achieved a test accuracy of 87.56% on CIFAR-10, benefiting from the integration of batch normalization, data augmentation, dropout, global average pooling, and label smoothing. ResNet-20 further improved results, reaching 90.64% on CIFAR-10 and 65.46% on CIFAR-100. Incorporating attention mechanisms led to additional performance gains, pushing test accuracy to 90.72% and 65.81%, respectively. These findings highlight the effectiveness of combining architectural enhancements with regularization techniques to construct high-performing, noise-resilient CNNs from scratch.

.

# 1 Introduction

Image classification is a core task in computer vision with applications ranging from medical diagnosis to autonomous driving. While CNNs have achieved state-of-the-art results, most rely on pre-trained models and large datasets. Training from scratch remains important in resource-constrained or domain-specific settings. This project investigates how architectural choices, regularization, and training strategies affect learning and generalization when building CNNs from scratch.

# 2 Related Work

VGG Architecture: Introduced in Karen Simonyan [2015], this architecture is built upon the use of small $3\times3$ convolutional filters, which enable the construction of deep and efficient networks. It marked a significant improvement over prior-art configurations by demonstrating that increasing depth using a stack of simple, uniform convolutional layers can substantially enhance performance.

Residual Networks: The landmark work Kaiming He et al. [2015] introduced deep residual learning, which enabled the successful training of very deep convolutional networks. The key innovation was the use of residual connections—also called identity mappings—that allow layers to be bypassed through shortcut paths. These connections preserve gradient flow during backpropagation and effectively address the degradation problem in deep networks. ResNet's design has since become foundational in modern CNN architectures.

Attention Mechanisms: Squeeze-and-Excitation (SE) blocks, introduced in Hu et al. [2019], enhance feature representations by modeling interdependencies between channels and adaptively recalibrating their responses. This allows the network to emphasize more relevant features during learning. CBAM, proposed in Woo et al. [2018] extends this idea by applying attention along both channel and spatial dimensions. Despite their lightweight design, both SE and CBAM can be easily integrated into existing architectures and have shown consistent improvements across various vision benchmarks.

ConvNeXt: proposed by Liu et al. [2022], modernizes convolutional networks by incorporating design ideas from Vision Transformers such as large kernels, inverted bottlenecks, and LayerNorm while retaining the computational efficiency of CNNs. It achieves competitive performance on the large-scale ImageNet-1K benchmark using only convolutional operations and now serves as a robust baseline for image classification tasks.

Symmetric Cross-Entropy (SCE): Wang et al. [2019] addresses this by combining the standard cross-entropy loss with a "reverse" cross-entropy term that penalizes overconfident fitting of potentially incorrect labels. Wang et al. demonstrates that SCE outperforms vanilla cross-entropy under high noise rates on CIFAR-10, making it an effective drop-in replacement for noisy-label scenarios.

# 3 Data

We use CIFAR-10 and CIFAR-100, two widely adopted image classification benchmarks consisting of 60,000 color images at 32×32 resolution. CIFAR-10 spans 10 object classes, while CIFAR-100 covers 100 fine-grained categories. Images are normalized per channel, with a fixed validation split of 5,000 training images. Data augmentation includes horizontal flips and random translations. For reference, Vision Transformer models such as ViT-H/14 Dosovitskiy et al. [2021] have achieved over 99% accuracy on CIFAR-10, while EfficientNet-L2 Mingxing Tan [2021] combined with Sharpness-Aware Minimization (SAM) Pierre Foret et al. [2021] reaches over 96% on CIFAR-100.

We also use ImageNette[1], a 10-class subset of ImageNet-1K, containing approximately 9,500 training and 4,000 validation images at 160px resolution. It allows faster benchmarking of modern architectures like ConvNeXt under ImageNet-style conditions, without the full computational cost. We apply standard preprocessing: random resized crops, horizontal flips, and ImageNet normalization.

[1] `https://github.com/fastai/imagenette`

## 4 Proposed Method

Our experiments build on a baseline CNN that we incrementally enhance through controlled ablations. First, each new architectural or regularization module is added in isolation to measure its individual effect; then, combinations of modules are evaluated to identify synergistic gains. To ensure fair comparisons, all runs use the same random seed. We split each dataset into a training set and a 5,000-sample validation hold-out, tuning our designs on validation performance before reporting final results on the 10,000-image test set. Throughout training, we log cross-entropy loss and top-1 accuracy for both train and validation, using their curves to diagnose learning dynamics and overfitting. All implementations use PyTorch.

### 4.1 Architectural & regularization investigations

#### 4.1.1 Squeeze-and-Excite layers

To enhance channel-wise feature representation, we integrate SE blocks into the ResNet20 architecture, following Hu et al. [2019]. Each SE block was inserted after the final convolution of a residual block and before the shortcut addition. It performs three steps:

- Squeeze: Global average pooling compresses each channel to a scalar,

$$z_c = \frac{1}{HXW} \sum_{i=1}^{H} \sum_{j=1}^{W} U_{i,j,c}$$

  We implemented this using `nn.AdaptiveAvgPool2d(1)`, which reduces the spatial dimensions of each feature map to $1 \times 1$ by averaging over height and width.
- Excitation: A two layer MLP with ReLU and sigmoid outputs scaling weights

$$s = \sigma((W_2 \cdot \delta(W_1 \cdot \mathbf{z}))$$

  with reduction r =8.
- Scale: Each channel is then rescaled: $\tilde{X}_c = s_c \cdot X_c$

#### 4.1.2 Upgrade to Squeeze-and-Excite: CBAM

To enhance both channel and spatial features, we integrate CBAM blocks into ResNet-20, following Woo et al. (2018). Each block is placed after the final convolution in a residual block and before the shortcut addition. CBAM applies attention in two steps:

- **Channel attention**: Global average and max pooling are applied across spatial dimensions. Both outputs are passed through a shared MLP with reduction ratio r=8. The results are summed and activated by a sigmoid to produce channel weights, which rescale the input feature map.
- **Spatial attention**: From the channel-attended output, average and max pooling are applied along the channel axis. The two maps are concatenated and passed through a 7×77×7 convolution and sigmoid to produce spatial attention, which is used to rescale the feature map again.

The final output is then passed to the residual path. This helps the network focus on both informative channels and spatial regions, with minimal overhead.

#### 4.1.3 ConvNeXt architecture

ConvNeXt Liu et al. [2022] modernizes traditional convolutional networks by incorporating a few key ideas from Vision Transformers while keeping the convolutional backbone intact. Its Tiny variant is defined by four stages of repeated blocks (depths [3,3,9,3]) with channel dimensions [96, 192, 384, 768]. Each stage features: **Patchify stem:** a single large-stride convolution that converts the image into non-overlapping patches. **Inverted bottleneck blocks:** depthwise convolutions with large kernels followed by a small "expansion–projection" multilayer perceptron and learnable channel scaling. **Channel-last normalization:** layer normalization applied in the spatially flattened format for stable training. **Downsampling layers:** simple convolutions that halve spatial resolution and double channels between stages. textbfStochastic depth: lightweight regularization that randomly skips blocks (up to 10% drop rate) during training.

A global average pooling and linear classification head complete the model. This streamlined design delivers Transformer-level accuracy on ImageNet-1K with minimal extra complexity.

### 4.2 Make training more robust to noisy labels

Real-world annotations often contain errors, so we evaluate our ConvNeXt-Tiny pipeline under synthetic label noise. We corrupt a fraction of the CIFAR10 training labels by randomly reassigning each selected ground-truth to one of the other classes. To mitigate overfitting to corrupted targets, we replace the standard Cross-Entropy (CE) loss with Symmetric Cross-Entropy (SCE) Wang et al. [2019], defined as

$$L_{\mathrm{SCE}} = \alpha \, L_{\mathrm{CE}} + \beta \, L_{\mathrm{RCE}},$$

where the reverse-CE term penalizes overconfident predictions on potentially incorrect labels. We adopt $\alpha = 0.1$ and $\beta = 1.0$ as in the original work.

## 5 Experiments

### 5.1 Building a VGG-based network from scratch

#### 5.1.1 Baseline VGG-based network

Starting from the Assignment 3 code-base we re-implemented the classifier with `torch.nn` layers and automatic differentiation. The first upgrade inserted a single VGG block—two $3\times3$ convolutions followed by max-pooling—between the patchify stem and the fully–connected head. Trained for 30 epochs with `AdamW` (learning-rate $1\times10^{-3}$) and a triangular–cyclic schedule, this model reached **69.59%** test accuracy. Stacking three VGG blocks (doubling the channel width after each downsampling) increased representational capacity without a prohibitive compute cost and lifted the score to **73.84 %**. Table 1 summarizes the final performance. In both cases, the network exhibits significant overfitting, with the training accuracy regularly reaching 100%, as illustrated in Figure 1.

Table 1: Test accuracies of the baseline VGG-based network for two different architectures.

| | Test Acc |
|---|---|
| 1 VGG Block | 69.59% |
| 3 VGG Blocks | 73.84% |

#### 5.1.2 Regularization

We next isolated the effect of individual regularisation techniques on the three-block baseline. Applying *dropout* ($p = 0.2$) regularly throughout the network improved generalisation slightly, whereas pure *weight decay* ($\lambda = 1\times10^{-3}$) had only a marginal effect. By contrast, lightweight *data augmentation*—random horizontal flips and $\pm4$-pixel translations—was highly effective, pushing performance to **82.73 %** and significantly reducing over-fitting. Finally, we applied a dropout rate that increased with network depth (from $p = 0.2$ to $p = 0.5$), in combination with data augmentation and batch normalization after every convolutional layer. This strategy enabled stable training for 40 epochs at a higher learning rate ($5\times10^{-3}$) without any signs of overfitting. The resulting model attained **85.89 %** test accuracy. The influence of each technique is visualised in Figure 2, and the corresponding accuracies are listed in Table 2.

#### 5.1.3 Extensions

The best-performing model (3 VGG Blocks + Batch Norm + Dropout + Data Augmentation) was then re-trained, again for 40 epochs, with further extensions. These included adding label smoothing with a smoothing factor of $\epsilon = 0.2$, experimenting with different learning rate schedulers, namely step decay and cosine annealing with restarts, and modifying the VGG architecture. For the architectural changes, we tried to perform downsampling by setting the stride of the second

Table 2: Test accuracies of the VGG-based network with multiple regularization methods.

| | Test Acc |
|---|---|
| 3 VGG Blocks + Dropout | 78.27% |
| 3 VGG Blocks + Weight Decay | 73.44% |
| 3 VGG Blocks + Data Augmentation | 82.73% |
| 3 VGG Blocks + Batch Norm + Dropout + Data Augmentation | 85.89% |

convolution in each VGG block to 2, removing the corresponding max-pooling layers. Finally, we replaced the fully connected classification head with a global average pooling (GAP) layer following the final convolution. Table 3 summarizes the test accuracies. Only label smoothing and GAP led to performance improvements, even reaching **87.56%** test accuracy when combined.

Table 3: Test accuracies of the regularized 3 VGG blocks network with multiple extensions.

| | Test Acc |
|---|---|
| Label Smoothing | 86.42% |
| Step LR | 83.14% |
| Cosine Restart LR | 83.73% |
| Convolutional Down Sampling | 84.89% |
| GAP Head | 87.30% |
| GAP Head + Label Smoothing | 87.56% |

## 5.2 Investigating various ConvNet extensions

### 5.2.1 Architectural & regularization investigations

- **Baseline ResNet architecture**

To further improve the training accuracy, a 20-layer ResNet architecture has been trained from scratch on CIFAR-10 for 60 epochs with a combination of data augmentation, label smoothing ($\epsilon = 0.1$), and weight decay ($\lambda = 1 \times 10^{-4}$). Three different training configurations were compared to assess the impact of the optimizer and learning rate scheduler on model performance, with the goal of identifying the most effective setup. The `SGD` optimizer is configured with a momentum of 0.9, while the `AdamW` optimizer uses its default settings. Each network employs "identity shortcuts", where zero-padding is applied to the identity mapping to ensure dimensional consistency, following Option A from Kaiming He et al. [2015]. We adopted He initialization He et al. [2015], particularly suited for ReLU-based networks. Figure 3 illustrates the learning curves while Table 4 reports the final test accuracies. The `SGD` with momentum optimizer showed clear benefits from the learning rate decay schedule, particularly after the first decay step (17th update), which led to a notable boost in performance. This improvement enabled `SGD` to outperform both `AdamW`-based configurations, achieving a final test accuracy of **90.64%**.

- **ResNet extensions**

Building upon the baseline `ResNet`, SE and CBAM attention modules were sequentially inserted in the architecture, aiming at further improving the network's representational capacity. These new architectures were trained from scratch on both CIFAR-10 and CIFAR-100 with the configuration

Table 4: Test accuracies of ResNet-20 on CIFAR-10 with different training configurations.

| | Test Acc |
|---|---|
| SGD with momentum + Step LR | 90.64% |
| Adam + Step LR | 89.39% |
| Adam + Cosine Restart LR | 89.83% |

that gave us the best performance on the validation set with the baseline `ResNet`: aggressive data augmentation, label smoothing, `SGD` optimizer with a 0.9 momentum, and a step decay scheduler. For the CBAM-enhanced networks, we experimented with two variants of the channel attention mechanism: one using only average pooling (equivalent to the SE module), and another combining both average and max pooling operations to enrich the attention signal. Figures 4 and 5 illustrate the learning curves, while Table 5 reports the achieved test accuracies. The inclusion of SE modules consistently led to improved performance on both datasets, with a more pronounced gain observed on CIFAR-100, a dataset known for its higher complexity and greater number of classes. The performance impact of the CBAM extension was more nuanced. The variant using only channel-wise average pooling provided a modest improvement over the baseline, but only on CIFAR-100. In contrast, the full CBAM configuration, employing both average and max pooling, consistently resulted in degraded performance across both datasets.

To further investigate the CBAM, the model was re-trained using a cosine–decayed RMSprop optimiser. The model achieved test accuracy within 80 epochs, matching the peak performance of `AdamW` while sidestepping the numerical instabilities we observed when scaling to larger backbones. Transferring the same training recipe to CIFAR–100 yielded 65.54% accuracy, suggesting that attention, rather than sheer depth, drives the improvement across datasets with more classes. Attempts to enlarge the architecture beyond `ResNet` offered little gain: deeper variants converged faster with `AdamW` but required heavier regularisation to curb overfitting, whereas the compact CBAM–ResNet provided the best accuracy–to–compute trade–off.

Table 5: Test accuracies of ResNet-20 and its extensions on CIFAR-10 and CIFAR-100.

| | CIFAR-10 | CIFAR-100 |
|---|---|---|
| ResNet-20 | 90.64% | 65.46% |
| ResNet-20 + SE | 90.72% | 65.81 |
| ResNet-20 + CBAM (Avg Pooling) | 90.42% | 65.54% |
| ResNet-20 + CBAM (Avg Pooling + Max Pooling) | 89.95% | 65.0% |

- **ConvNeXt**

We benchmark ConvNeXt-Tiny on the ImageNette 160px subset to approximate ImageNet-1K performance. The hyperparameters used were the same as the ones used in the original paper. Table 6 compares our results to the original work.

| Model | Top-1 Acc (%) | Origin |
|---|---|---|
| ConvNeXt-Tiny (original - IN-1k) | 84.10 | from Liu et al. [2022] |
| ConvNeXt-Large (original - IN-1k) | 86.60 | from Liu et al. [2022] |
| ConvNeXt-Tiny (ours - ImageNette-10) | 86.42 | this work |

Table 6: ImageNette 160px benchmark results.

In figure 6, it can be seen that our results were successfully able to reproduce those of the papers. The improvement over the original paper may be caused by a difference in dataset between ImageNette and ImageNet 1k. ImageNet 1k was not used due to computational limitations.

### 5.3 Make training more robust to noisy labels

In figure 7 we compare Cross-Entropy (CE) versus Symmetric Cross-Entropy (SCE, $\alpha = 0.1, \beta = 1.0$) under 40% symmetric label noise on CIFAR-10. All experiments use the same ConvNeXt backbone, AdamW, standard random-crop + horizontal-flip augmentation. Although the same hyperparameters as the original paper were used, the results of the paper were not able to be replicated. By using the same hyperparameters as the initial paper, the CE model outperformed the SCE model by around 9%. The SCE model achieves 70.5% and the CE model achieves 79.37%. Other hyperparameter setups were explored, and some of them got close to reproducing the paper's results. However, none of the experiments could quite match it. In one of the training runs, the model using SCE outperformed the model using CE by more than a percent accuracy on the test set, and the

CE model greatly overfit on the training data, as can be seen in 7. However, on multiple other runs, the CE model outperformed the SCE model. These results suggest either that there was an error in the implementation or that the results that the paper claims do not generalize as well as the authors claim, and that other factors may be at play other than just the loss metric, such as network architecture.

# 6 Conclusion

In summary, our experiments show that careful architectural design, regularization, and attention to loss functions can push shallow CNNs like ResNet-20 beyond 90% accuracy on CIFAR-10, with SE modules offering the best trade-off between performance and simplicity. Meanwhile, ConvNeXt-Tiny achieved over 86% on ImageNette, highlighting how re-engineered ConvNet blocks alone can rival transformer-level performance even on small-scale benchmarks.

## 6.1 Key Learnings

Throughout this project we gained practical and theoretical insights into Modern Convolutional neural network training pipelines:

- **Regularization with Data Augmentation:** Deeper VGG models easily overfit, achieving 100% training accuracy but poor generalization. Simple augmentations like flips and shifts, combined with dropout and batch norm, significantly improved stability and test performance.
- **Architectural Shift with ResNet:** Residual connections enabled deeper networks to train stably by preserving gradient flow and preventing information loss. This architectural change alone helped reduce overfitting and improved convergence.
- **Impact of Attention Modules:** SE and CBAM added to ResNet provided modest gains without destabilizing training. Their limited impact suggests that in already well-optimized, shallow networks on simple datasets, attention modules offer diminishing returns.
  The CBAM variant using only average pooling performed comparably to the baseline ResNet (90.42% vs 90.64%), while the full CBAM with both average and max pooling slightly degraded performance to 89.95%, suggesting that while attention mechanisms enhance feature representation, their benefit depends on the network's depth and the pooling strategy.
- **Convergence Speed vs. Stability:** AdamW led to faster convergence in shallow models like VGG but was unstable in deeper ones. SGD with StepLR offered slower but more stable and generalizable training, making it more suitable for deep architectures.
- **Modern ConvNet with ConvNeXt:** Adopting the ConvNeXt-Tiny backbone delivered transformer-level accuracy on ImageNette ( 80 %+), confirming that careful re-engineering of ConvNet blocks (large kernels, inverted bottlenecks, channel-last normalization, stochastic depth) can match more complex architectures with minimal overhead.
- **Robust Loss for Noisy Labels:** Symmetric Cross-Entropy (SCE) provided mitigated results under 40 % label noise. The implementation should be reexamined and other avenues explored to improve training under noisy labels.

## 6.2 Future ideas

**Scaling ConvNeXt to Full ImageNet**: Building on these findings, it would be pertinent to explore scaling ConvNeXt to larger variants and full ImageNet-1K to validate transfer to high-resolution settings with more classes.

**Scaling to Deeper Networks**: SE and CBAM showed modest gains on ResNet-20; applying them to deeper models like ResNet-50 or WideResNet could better highlight their benefits, especially on harder datasets like CIFAR-100 or ImageNet.

# References


Alexey Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv*, 2021.

Kaiming He et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv*, 2015.

Jie Hu et al. Squeeze-and-excitation networks. *arXiv*, 2019.

Xiangyu Zhang Kaiming He et al. Deep residual learning for image recognition. *arXiv*, 2015.

Andrew Zisserman Karen Simonyan. Very deep convolutional networks for large scale image recognition. *arXiv*, 2015.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11966–11976, 2022. doi: 10.1109/CVPR52688.2022.01167.

Quoc V. Le Mingxing Tan. Efficientnetv2: Smaller models and faster training. *arXiv*, 2021.

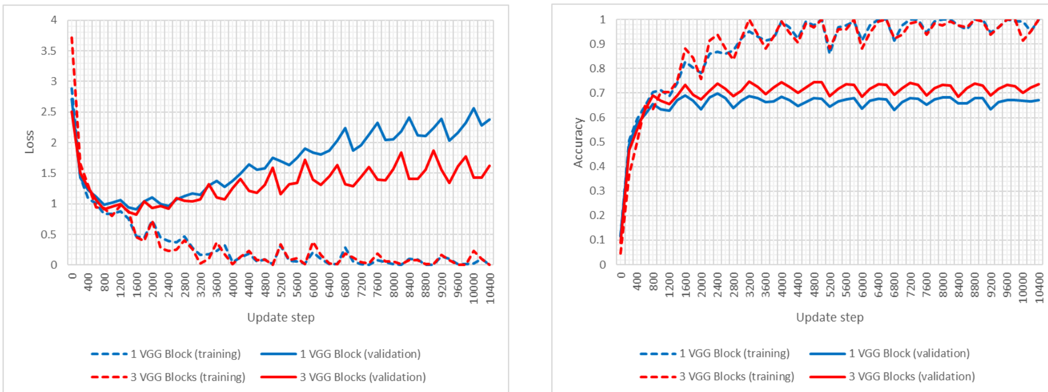
Ariel Kleiner Pierre Foret et al. Sharpness-aware minimization for efficiently improving generalization. *arXiv*, 2021.

Yisen Wang, Xingjun Ma, Zaiyi Chen, Yuan Luo, Jinfeng Yi, and James Bailey. Symmetric cross entropy for robust learning with noisy labels, 2019. URL `https://arxiv.org/abs/1908.06112`.

Sanghyun Woo et al. Cbam: Convolutional block attention module. *arXiv*, 2018.

# 7 Appendix

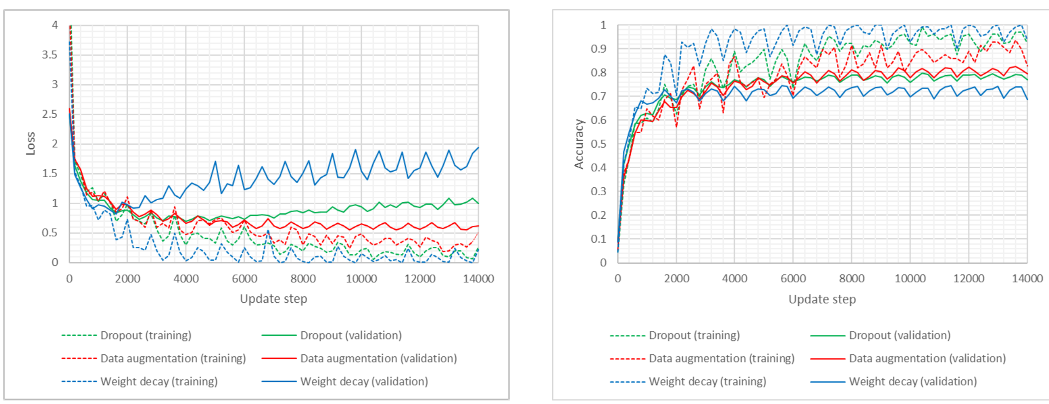
## 7.1 Baseline VGG




(a) Loss (b) Accuracy

Figure 1: Loss and Accuracy of the baseline VGG-based network for two different architectures.

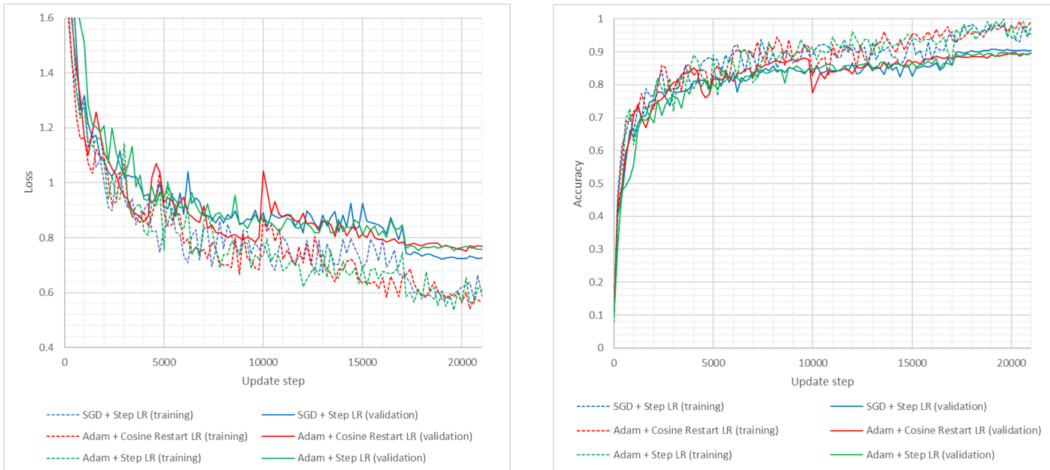
## 7.2 Regularization




(a) Loss (b) Accuracy

Figure 2: Loss and Accuracy on both training and validation sets for different regularization methods.

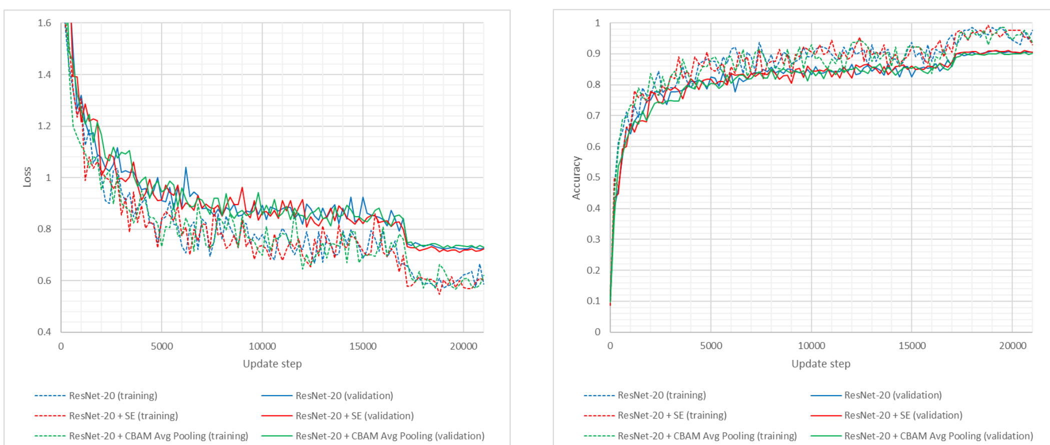
## 7.3 Baseline Resnet Architecture




(a) Loss (b) Accuracy

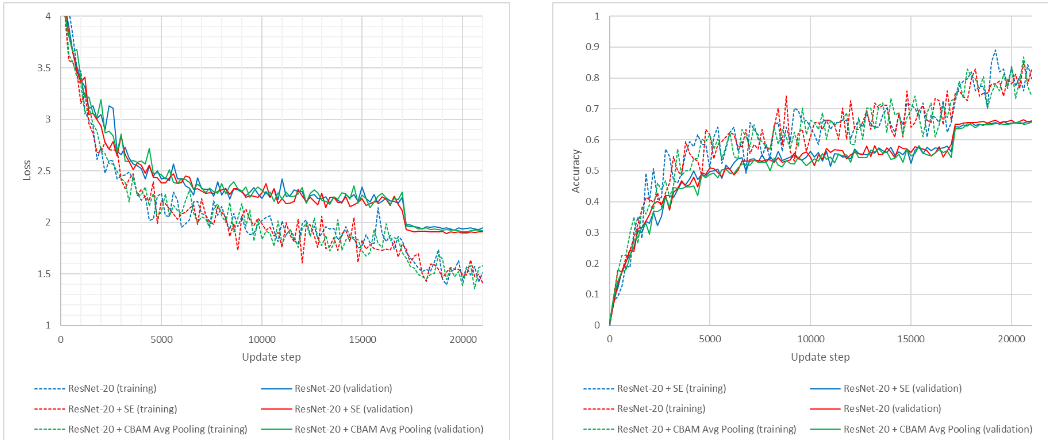
Figure 3: Loss and Accuracy of ResNet-20 on CIFAR-10 with different training configurations.

## 7.4 Resnet Extensions




(a) Loss (b) Accuracy

Figure 4: Loss and Accuracy of ResNet-20 and its extensions on CIFAR-10.




(a) Loss

(b) Accuracy

Figure 5: Loss and Accuracy of ResNet-20 and its extensions on CIFAR-100.

## 7.5 ConvNeXt

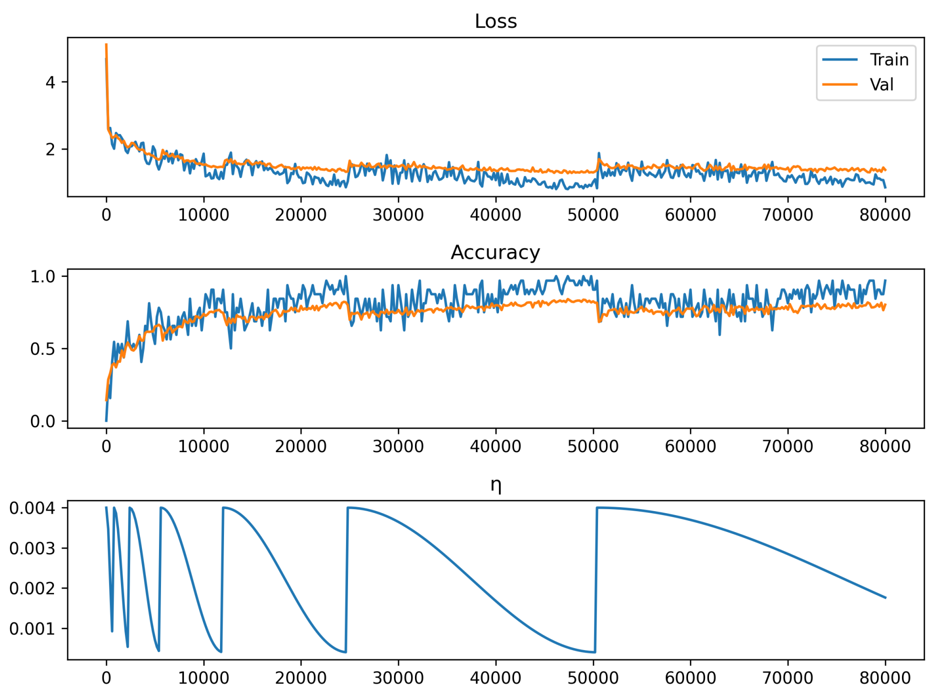



Figure 6: Training curves for the ConvNeXt with the original work's hyperparameters on CIFAR-10. **Top:** Loss on training (blue) and validation (orange); **Middle:** Accuracy; **Bottom:** Learning rate schedule (CyclicLR).

### 7.6 Make training more robust to noisy labels

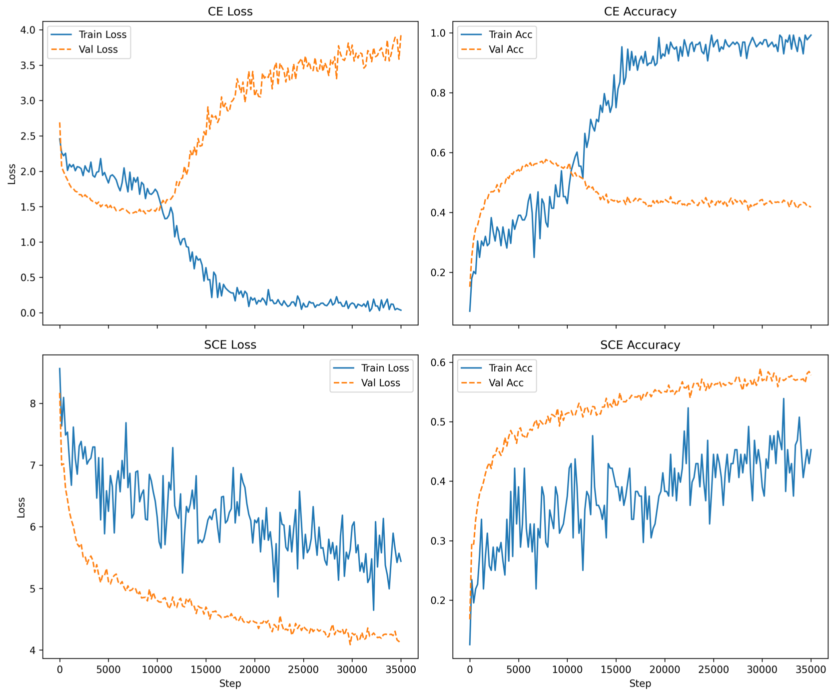



Figure 7: Comparison of cross-entropy (CE) vs symmetric cross-entropy (SCE).**Top row:** CE loss (left) and accuracy (right).**Bottom row:** SCE loss (left) and accuracy (right).