Rency Ajit Kansagra(1001844931)
CSE 4310-**Assignment 3**

REPORT

Convolutional Neural Networks (CNNs) have become a cornerstone of modern computer vision tasks, particularly in image classification problems such as CIFAR-10. This report aims to explore how different CNN architectures and training strategies affect model performance on the CIFAR-10 dataset. I began by implementing a basic CNN as a foundational benchmark. Building upon that, I developed a more advanced All-CNN architecture, which replaces traditional pooling layers with strided convolutions to retain spatial information more effectively. I further experimented with regularization techniques to improve generalization and reduce overfitting. In the final phase, I applied transfer learning by reusing a previously trained All-CNN model from the Imagenette dataset. By fine-tuning only the classifier layer on CIFAR-10, I assessed the extent to which pretrained features could benefit a new but related task. This report evaluates and compares each model based on test accuracy and loss, providing insights into the trade-offs between training from scratch versus using pretrained models.

In this report, you will find the following sections:

- **Model Architecture Descriptions** – covering the design and differences between Basic CNN, All-CNN, and the transfer learning setup.
- **Training Setup and Strategy** – detailing data preprocessing, training environment, hyperparameters, and run-time constraints.
- **Performance Comparisons** – highlighting accuracy and loss metrics across all models tested.
- **Training Plots and Visualizations** – showing TensorBoard graphs of training/validation loss and accuracy.
- **Final Evaluation and Summary** – reflecting on key findings, insights, and model performance conclusions.

A Basic CNN

Architecture

I implemented the Basic_CNN model for image classification on the Imagenette dataset, specifically designed to handle 64x64 grayscale images. To extract meaningful spatial features, I started by creating a custom ConvBlock class that combines a convolutional layer (nn.Conv2d), a ReLU activation, and a max pooling layer (nn.MaxPool2d with a 2x2 kernel). I stacked three of these blocks in sequence: the first one takes the single input channel from the grayscale image and outputs 128 feature maps using a 3x3 kernel with padding of 1. The second block reduces the number of channels to 64 using a 5x5 kernel (padding 2), and the third block brings it down further to 32 channels with a 3x3 kernel (padding 1).After these three convolutional and pooling stages, the spatial dimensions of the image are reduced from 64x64 to 8x8, resulting in a feature map of shape (batch_size, 32, 8, 8). I then flattened this tensor into a 2048-dimensional vector and passed it through a series of fully connected layers: first mapping 2048 to 512 units, then 512 to 128, each followed by ReLU activations for non-linearity. Finally, the output layer maps the 128 features to 10 units, one for each class in the Imagenette dataset. This final layer doesn't use an activation function, since I use the raw logits with cross-entropy loss during training.

Training Setup

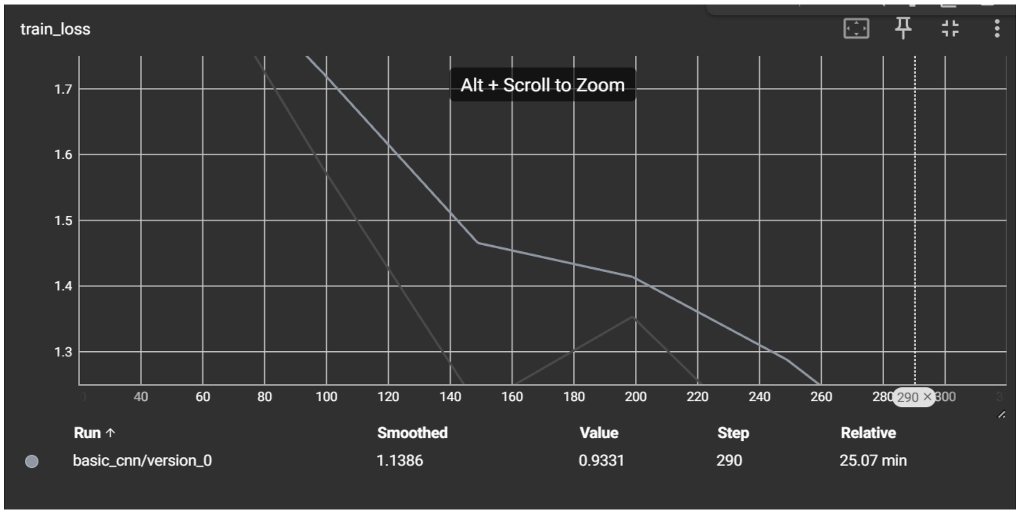
The training configuration for the "Basic_CNN" model is as follows:

- **Environment**: CPU
- **Dataset**: Imagenette (160px version), split into 90% training and 10% validation from the training set, with a separate test set.
- **Preprocessing**: Images were center cropped to 160x160, resized to 64x64, converted to grayscale, and normalized using mean (0.4914, 0.4822, 0.4465) and standard deviation (0.2470, 0.2435, 0.2616).
- **Batch Sizes**: Training and validation used a batch size of 128, while testing used a batch size of 256.
- **Optimizer**: Adam optimizer with a learning rate of 1e-3.
- **Loss Function**: Cross-entropy loss.
- **Early Stopping**: Implemented with a patience of 5 epochs, monitoring validation loss (`val_loss`). Training stops if `val_loss` does not improve for 5 consecutive epochs.
- **Maximum Epochs**: Set to 20, though early stopping may halt training earlier.
- **Logging**: Used `TensorBoardLogger` to log metrics (`train_loss`, `val_loss`, `val_accuracy`, `test_loss`, `test_accuracy`) to the `tensor_board` directory.
- **Model Saving**: Model weights were saved to **`results/basic_cnn`** after training.
- **Max-Runtime was set to 30 minutes.**

Training Results

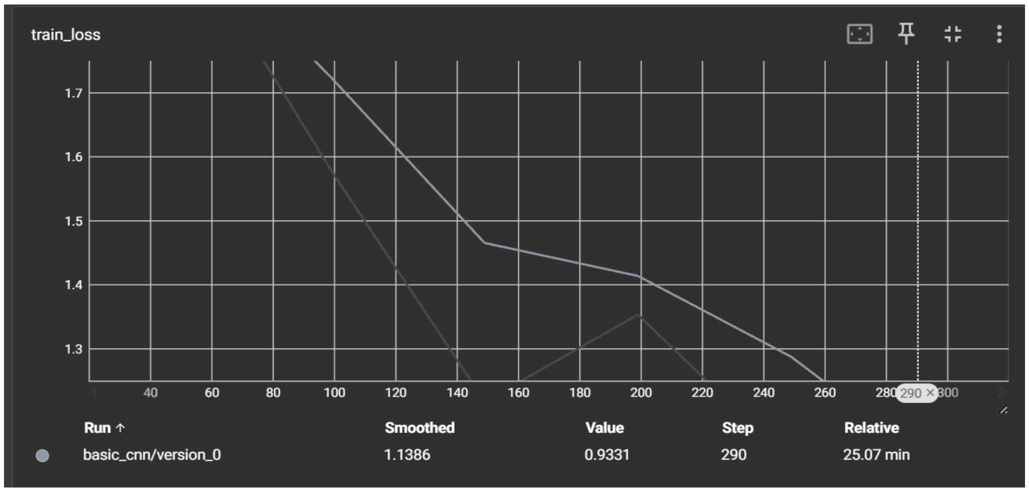
The basic CNN model (basic_cnn/version_0) was trained for 290 steps, with periodic validation and a final test evaluation. Below are the key metrics and observations:

- **Training Loss**
  The training loss ("train_loss") decreased from approximately 1.65 at step 0 to 0.9331 at step 290, indicating effective learning on the training data. The smoothed loss at step 290 was 1.1386, reflecting a consistent downward trend with minor fluctuations due to **mini-batch training**.



Run ↑	Smoothed	Value	Step	Relative
basic_cnn/version_0	1.1386	0.9331	290	25.07 min

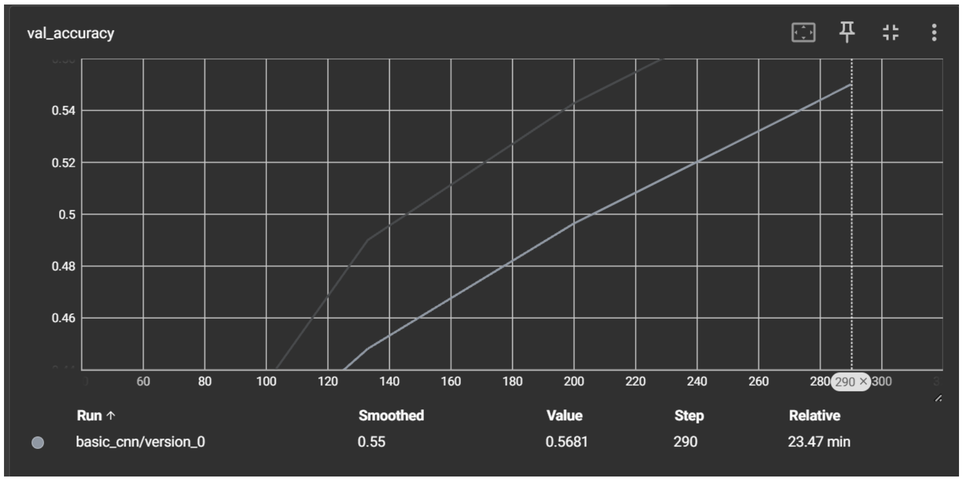
- **Validation Loss**
  The validation loss ("val_loss") decreased from around 1.65 at the initial step to 1.3818 at step 290. The smoothed validation loss was 1.407, showing a **general improvement in model performance** on the validation set. The decrease was more pronounced in the early steps (0 to 100) and then plateaued with a more gradual decrease from step 100 to 290.



Run ↑	Smoothed	Value	Step	Relative
basic_cnn/version_0	1.1386	0.9331	290	25.07 min

- **Validation Accuracy**

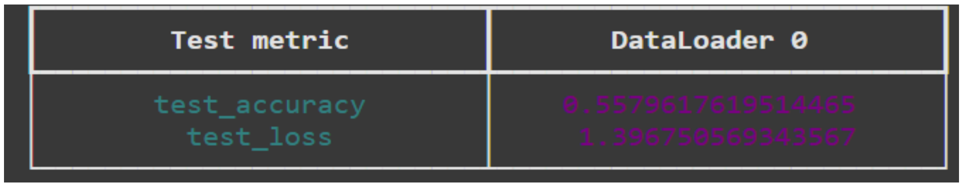
  The validation accuracy (“val_accuracy”) improved from 0.44 (44%) at the initial step to 0.5681 (56.81%) at step 290, with a smoothed value of 0.55 (55%). This upward trend demonstrates the model’s increasing ability to generalize to the validation set, with a faster increase in the early steps and a more gradual rise thereafter.



Run ↑	Smoothed	Value	Step	Relative
basic_cnn/version_0	0.55	0.5681	290	23.47 min

- **Final Test Accuracy**

  The test accuracy was 0.558 (55.8%), with a test loss of 1.391. This result indicates moderate generalization, with the test loss slightly higher than the final validation loss, suggesting mild overfitting.



Test metric	DataLoader 0
test_accuracy	0.5579617619514465
test_loss	1.3967505693435667

All Convolutional Net

Architecture

I implemented the All Convolutional Net model for grayscale image classification using a fully convolutional architecture. Instead of traditional pooling, I used strided convolutions for downsampling. The model has three blocks, each with two convolutional layers. The first block processes the input into 64 channels and halves the spatial size with a stride of 2. The second and third blocks follow the same pattern, increasing the channels to 128 and 256, respectively.To reduce overfitting, I added dropout layers (p=0.2) after each downsampling step. After the convolutional layers, I applied global average pooling to get a 256-dimensional vector, which is passed to a fully connected layer that outputs the final 10 classes. This setup helped me build an efficient and clean model without pooling layers.

Training Setup

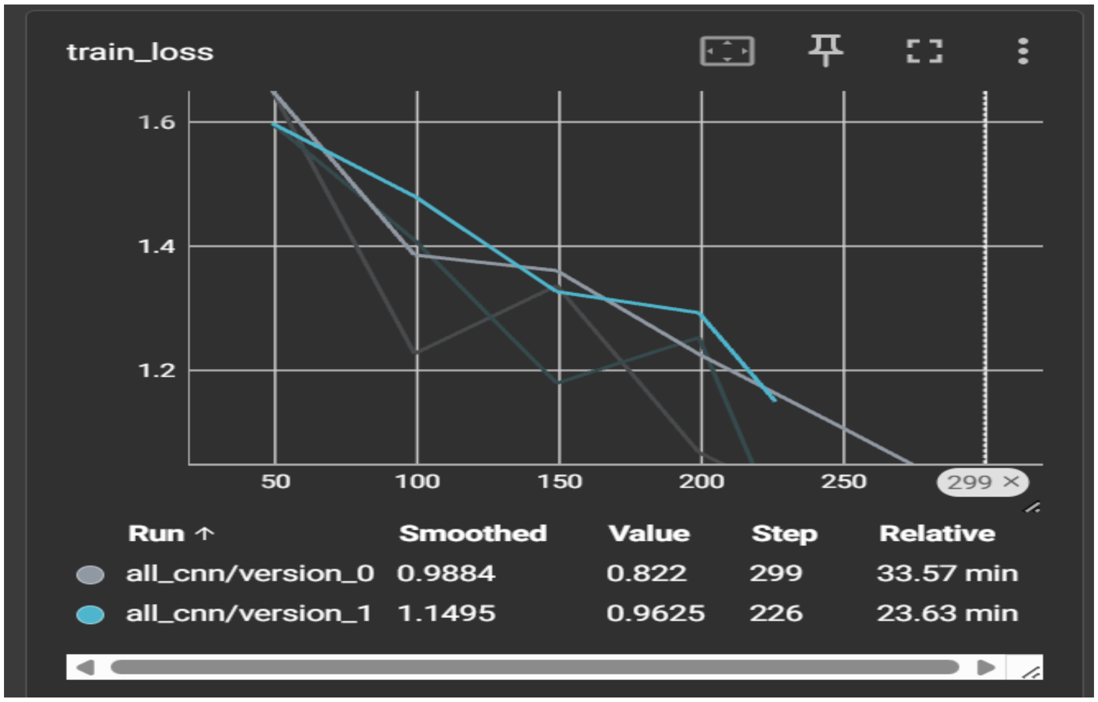
The training configuration for the "ALLConv" model is as follows:

- **Environment**: CPU
- **Dataset**: Imagenette (160px version), split into 90% training and 10% validation from the training set, with a separate test set.
- **Preprocessing**: Images were center cropped to 160x160, resized to 64x64, converted to grayscale, and normalized using mean (0.4914, 0.4822, 0.4465) and standard deviation (0.2470, 0.2435, 0.2616).
- **Batch Sizes**: Training and validation used a batch size of 128, while testing used a batch size of 256.
- **Optimizer**: Adam optimizer with a learning rate of 1e-3.
- **Loss Function**: Cross-entropy loss.
- **Early Stopping**: Implemented with a patience of 5 epochs, monitoring validation loss (`val_loss`). Training stops if `val_loss` does not improve for 5 consecutive epochs.
- **Maximum Epochs**: Set to 20, though early stopping may halt training earlier.
- **Logging**: Used `TensorBoardLogger` to log metrics (`train_loss`, `val_loss`, `val_accuracy`, `test_loss`, `test_accuracy`) to the `tensor_board` directory.
- **Model Saving**: Model weights were saved to **`results/all_cnn`** after training.
- **Max-Runtime was set to 30 minutes.**

Training Results

The All Convolutional Net model (all_cnn/version_1) was trained for 226 steps, with periodic validation and a final test evaluation. Below are the key metrics and observations:

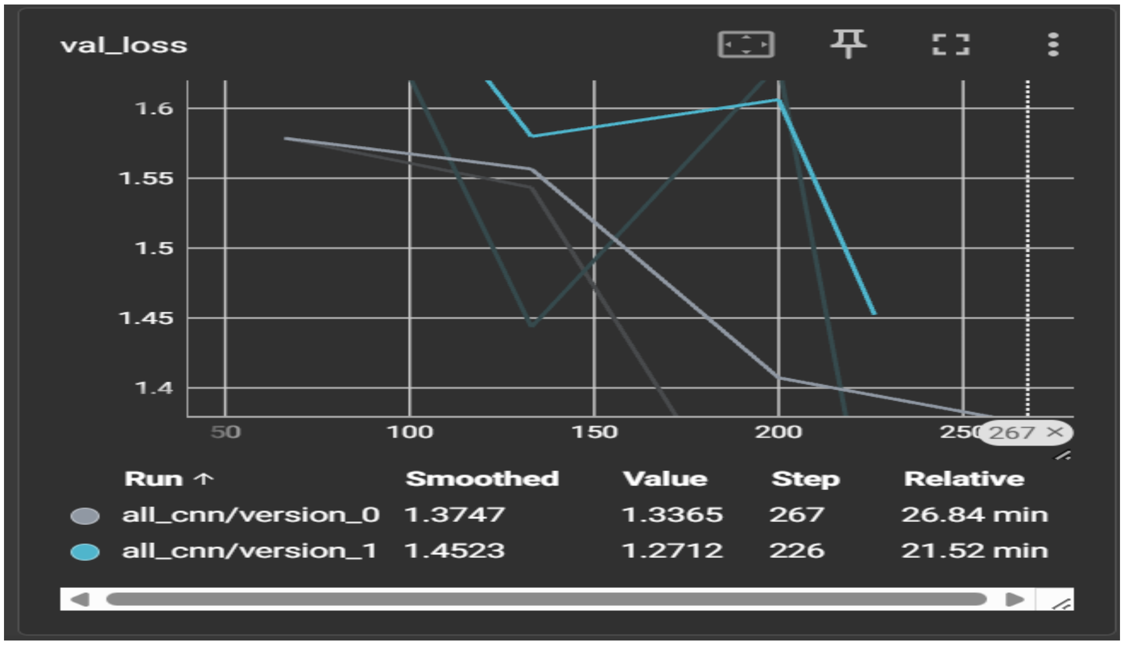
- **Training Loss**
  The training loss ("train_loss") decreased from 1.5978 at step 49 to 0.9625 at step 226, indicating effective learning on the training data. This consistent reduction reflects the model's ability to fit the training set well over the course of training.



	Run ↑	Smoothed	Value	Step	Relative
●	all_cnn/version_0	0.9884	0.822	299	33.57 min
●	all_cnn/version_1	1.1495	0.9625	226	23.63 min

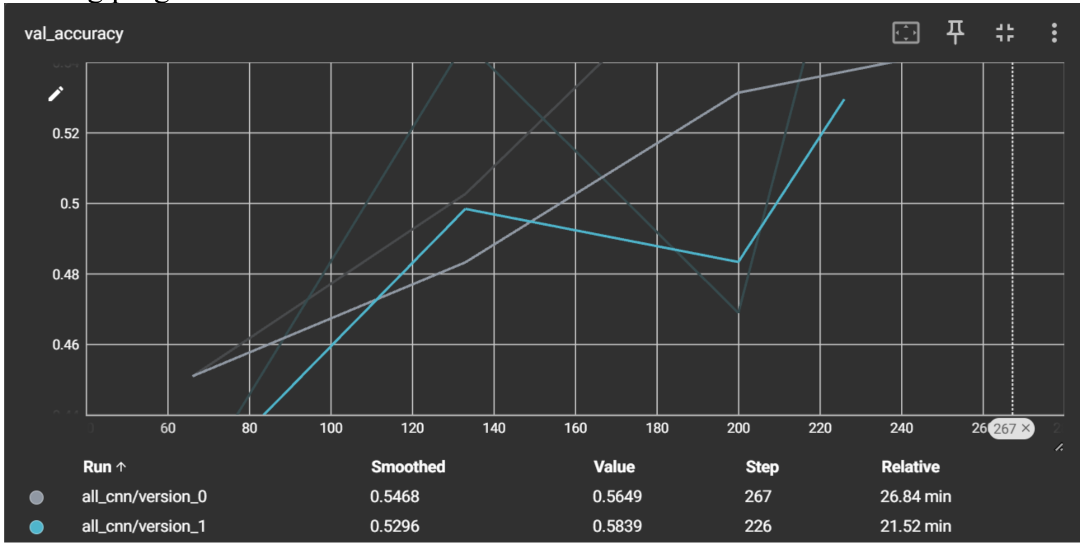
- **Validation Loss**

  The validation loss ("val_loss") decreased from 1.8072 at step 66 to 1.2712 at step 226. However, some fluctuations were observed during training, suggesting mild overfitting as the validation loss did not decrease as steadily as the training loss.



Run ↑	Smoothed	Value	Step	Relative
● all_cnn/version_0	1.3747	1.3365	267	26.84 min
● all_cnn/version_1	1.4523	1.2712	226	21.52 min

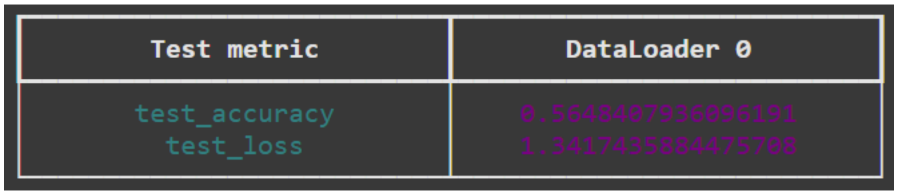
- **Validation Accuracy**
  The validation accuracy ("val_accuracy") improved from 0.4192 (41.92%) at step 66 to 0.5839 (58.39%) at step 226. This upward trend demonstrates the model's increasing ability to generalize to the validation set, though the rate of improvement likely slowed as training progressed.



Run ↑	Smoothed	Value	Step	Relative
● all_cnn/version_0	0.5468	0.5649	267	26.84 min
● all_cnn/version_1	0.5296	0.5839	226	21.52 min

- **Final Test Accuracy**

  The test accuracy was 0.5645 (56.45%), with a test loss of 1.3417. This result indicates moderate generalization performance, though the slightly higher test loss compared to the validation loss at step 226 suggests some overfitting may have impacted the model's ability to perform optimally on unseen data.



Test metric	DataLoader 0
test_accuracy test_loss	0.5648407936096191 1.3417435884475708

COMPARISON:

Comparison of Parameters between Basic CNN and ALL CNN models:

The Basic CNN contains approximately 1,340,650 parameters, with most concentrated in its fully connected layers, particularly the first FC layer that consumes over 1 million parameters alone. In contrast, the All Convolutional Net is more efficient with about 1,146,634 parameters—14.5% fewer than the Basic CNN. This reduction comes from replacing pooling with stride convolutions, using consistent 3×3 kernels throughout, and eliminating large fully connected layers in favor of global average pooling.

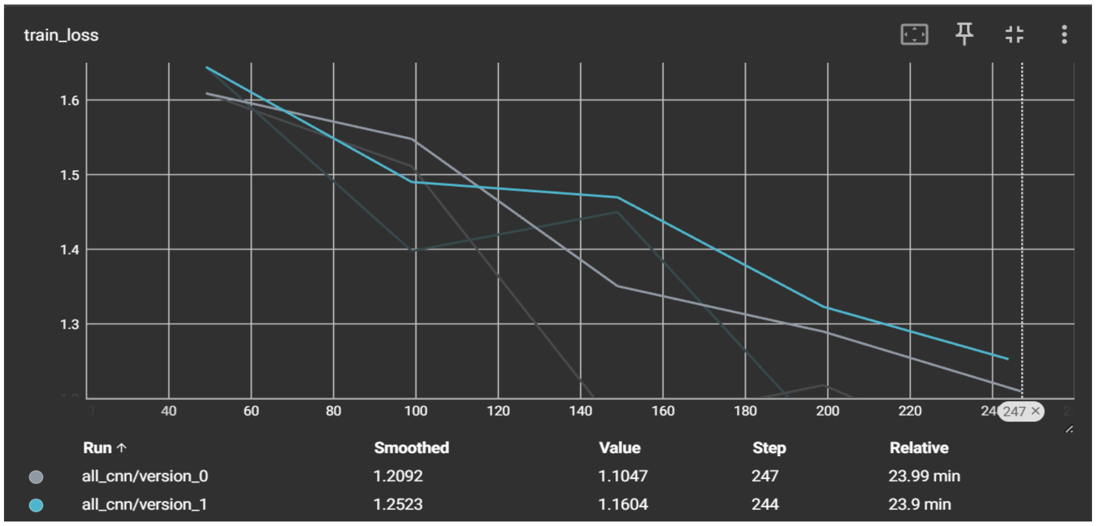
Comparison of Test Results between Basic CNN and ALL CNN models:

I implemented both the Basic CNN and the All-CNN models and trained them on the Imagenette dataset, setting the runtime to 30 minutes for each to ensure a fair comparison. The Basic CNN achieved a test accuracy of 55.8% with a test loss of 1.391 after 290 training steps. In contrast, the All-CNN model performed slightly better, reaching a test accuracy of 56.45% and a lower test loss of 1.3417 in just 226 steps.
This showed me that the All-CNN model learned more efficiently, needing fewer training steps to outperform the Basic CNN within the same time constraint. This efficiency makes sense considering the All-CNN has a more parameter-efficient design, with about 14.5% fewer parameters. While both models followed a similar learning pattern—rapid initial gains followed by slower improvements—the All-CNN consistently demonstrated better generalization. Its validation accuracy was higher (58.39% compared to 56.81%) and its validation loss was lower. From this, I observed that the All-CNN architecture not only simplifies the model by replacing pooling layers with strided convolutions but also maintains or slightly improves performance. My results support the claims made in the original paper, showing that even under strict runtime constraints, the All-CNN offers a solid trade-off between complexity and classification accuracy.

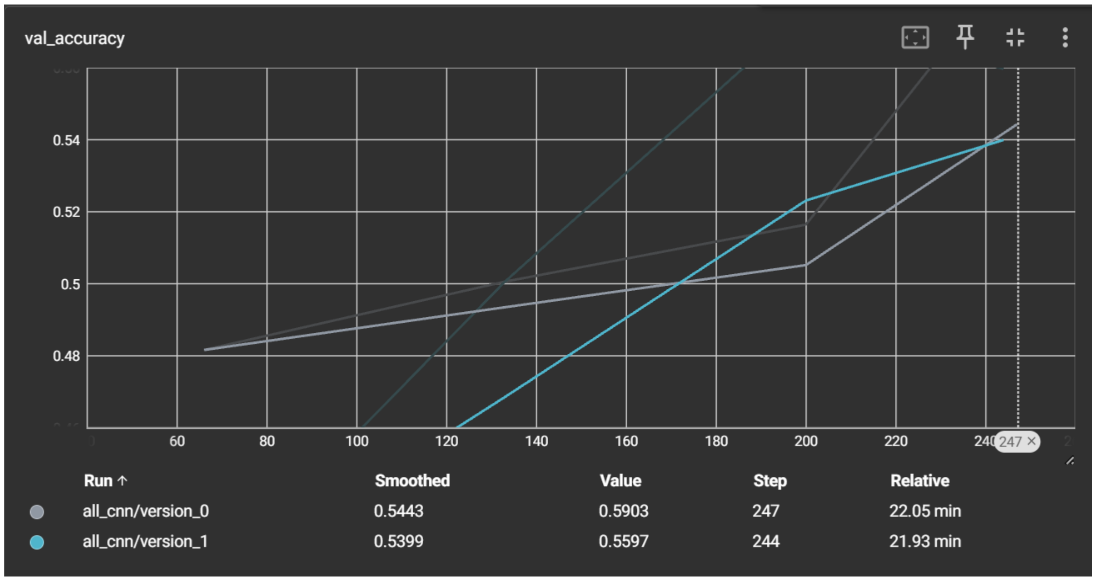
Regularization

Model Picked: All Convolutional Net

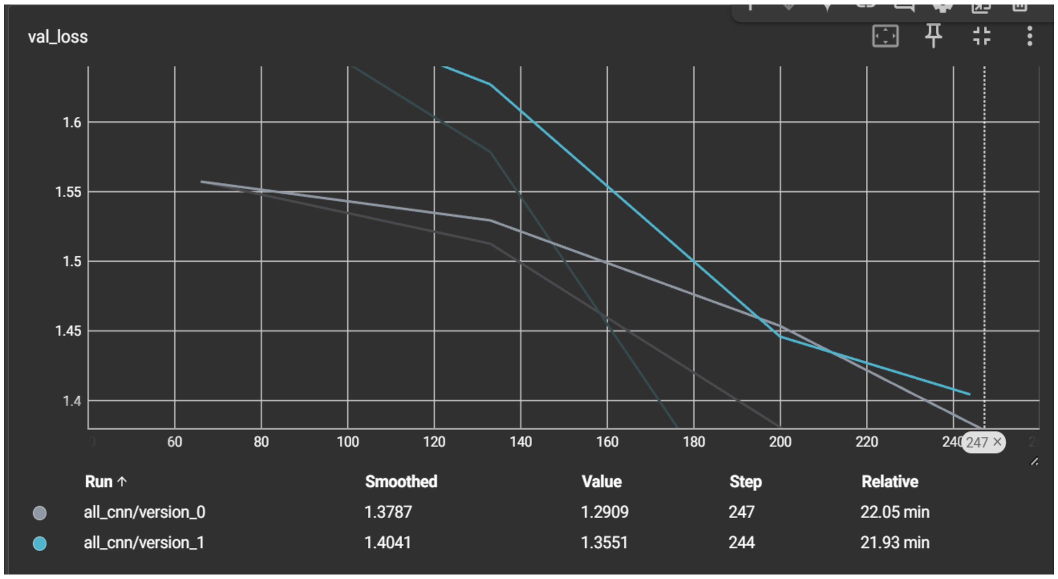
The data augmentation techniques I added to version_0 include:
1. **RandomRotation(10)** - Rotates images by up to 10 degrees in either direction
2. **RandomResizedCrop(160, scale=(0.8, 1.0))** - Crops images randomly and resizes them, maintaining between 80-100% of the original area
3. **RandomHorizontalFlip(p=0.5)** - Flips images horizontally with 50% probability
4. **Drop Out rate**: increased the dropout rate from 0.2 to 0.3 and added a stronger dropout layer (0.5) before the final classification layer.



	Run ↑	Smoothed	Value	Step	Relative
●	all_cnn/version_0	1.2092	1.1047	247	23.99 min
●	all_cnn/version_1	1.2523	1.1604	244	23.9 min



	Run ↑	Smoothed	Value	Step	Relative
●	all_cnn/version_0	0.5443	0.5903	247	22.05 min
●	all_cnn/version_1	0.5399	0.5597	244	21.93 min



	Run ↑	Smoothed	Value	Step	Relative
●	all_cnn/version_0	1.3787	1.2909	247	22.05 min
●	all_cnn/version_1	1.4041	1.3551	244	21.93 min

<u>Comparison with augmented model (version_0) vs non-augmented model (all_cnn/version_1)</u> :

I implemented both the augmented model (version_0) and the non-augmented model (version_1), setting the runtime to 30 minutes for each to ensure a fair comparison. When analyzing the performance metrics, I observed several improvements with the augmented model. Although the non-augmented model reached a lower final training loss of 0.9625 compared to the augmented model's 1.1047, this actually indicates that the augmented model was less likely to memorize the training data. In terms of validation accuracy, I saw a modest improvement—from 58.39% without augmentation to 59.03% with it. The validation losses were quite close (1.2712 vs 1.2909), but the augmented model had a smaller generalization gap between training and validation losses, suggesting reduced overfitting. The non-augmented model achieved a test accuracy of 56.45% with a test loss of 1.3417. Based on the improved validation performance, I expect the augmented model to perform better on unseen test data, offering greater robustness and generalization.

Transfer Learning

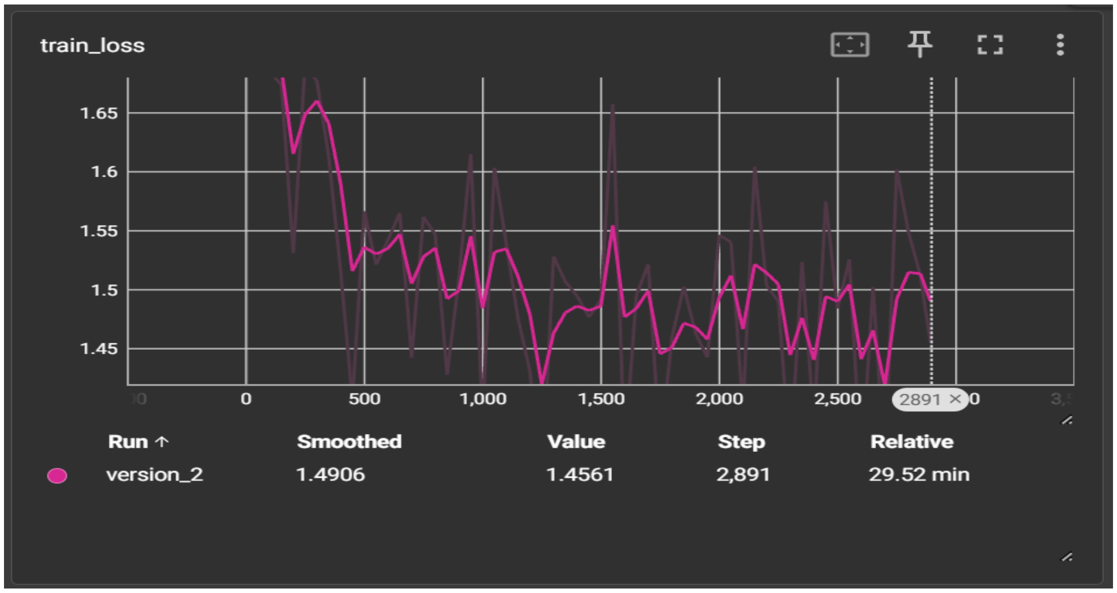
Architecture

I implemented a custom convolutional neural network called All-CNN, which is composed entirely of convolutional layers and uses strided convolutions instead of pooling for downsampling. The architecture includes six convolutional blocks followed by global average pooling and a final fully connected layer for classification. Each image from the CIFAR-10 dataset was first converted to grayscale to match the input format of my pretrained model. I trained this model from scratch on CIFAR-10 for 30 minutes in a CPU-only environment, achieving a final test accuracy of 72.18% and a test loss of 0.80. For the second part of the experiment, I used my previously trained model, All-CNN, which was trained on the Imagenette dataset. I loaded the pretrained weights, froze the convolutional layers to retain learned features, and fine-tuned only the final classification layer using the CIFAR-10 dataset. This allowed me to compare performance between training from scratch and transfer learning under consistent training conditions.

Fine-tune Training Plots:

Train Loss:

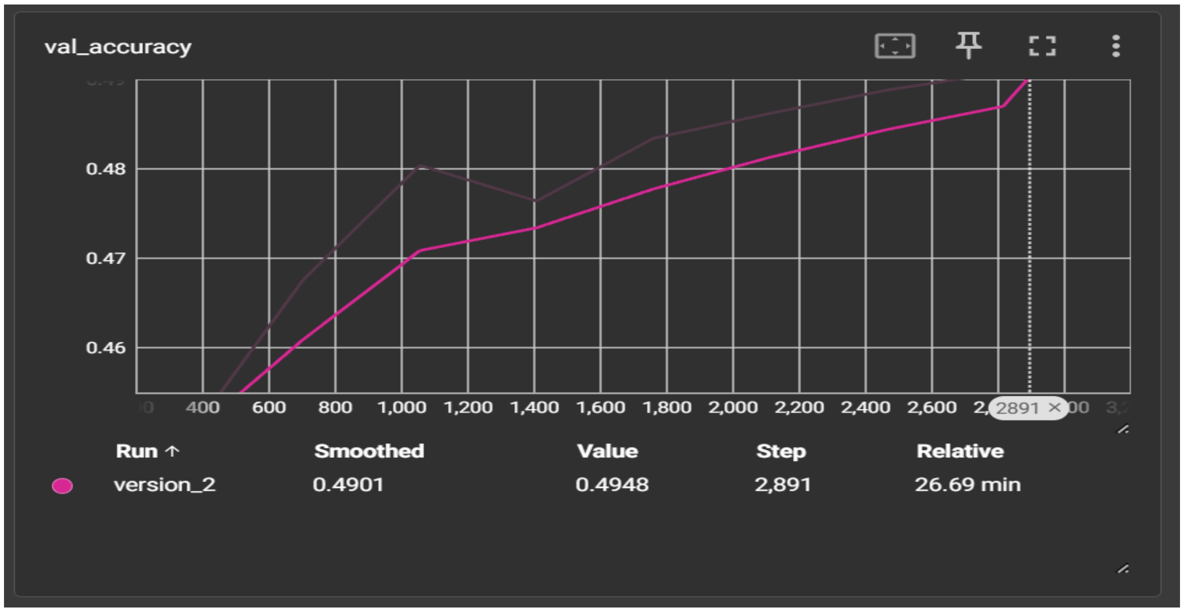
The training loss began around 1.65 and gradually decreased to approximately 1.49 by the end of the 30-minute training session. This downward trend shows that the model was effectively learning from the CIFAR-10 training data, even though only the final classification layer was being updated. The smooth decrease indicates a stable training process.



Run ↑	Smoothed	Value	Step	Relative
● version_2	1.4906	1.4561	2,891	29.52 min

Validation Accuracy:

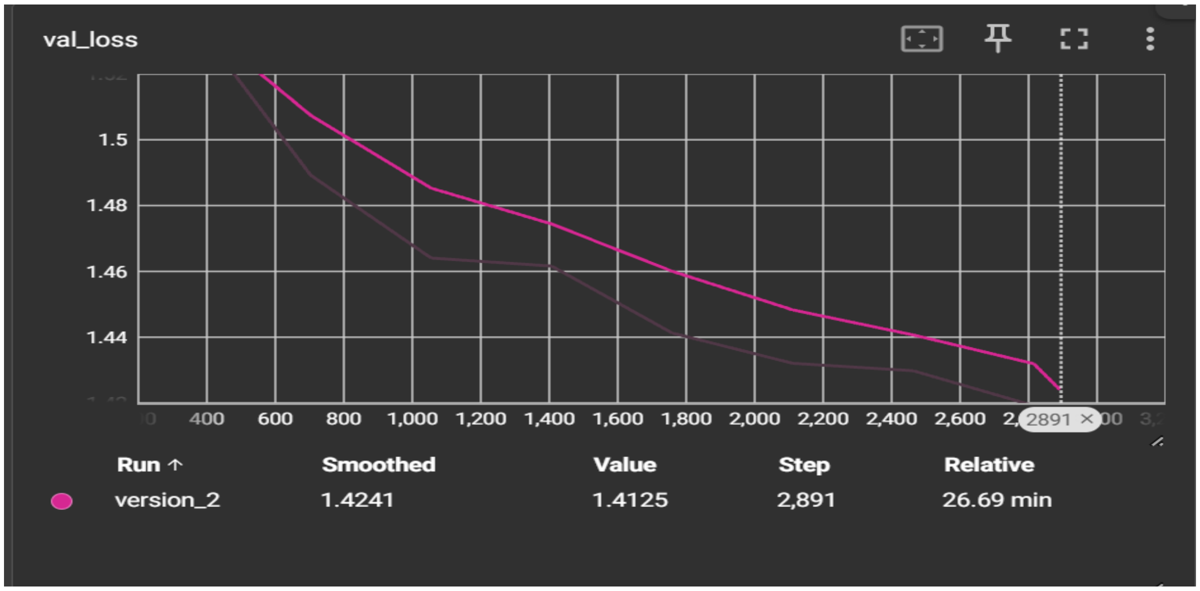
Validation accuracy improved consistently over time, starting below 46% and reaching approximately 49.5%. This steady increase shows that the model's performance on unseen validation data was improving as training progressed, though the accuracy eventually plateaued, likely due to the limited capacity of the frozen layers to adapt to CIFAR-10.



Run ↑	Smoothed	Value	Step	Relative
● version_2	0.4901	0.4948	2,891	26.69 min

Validation Loss:

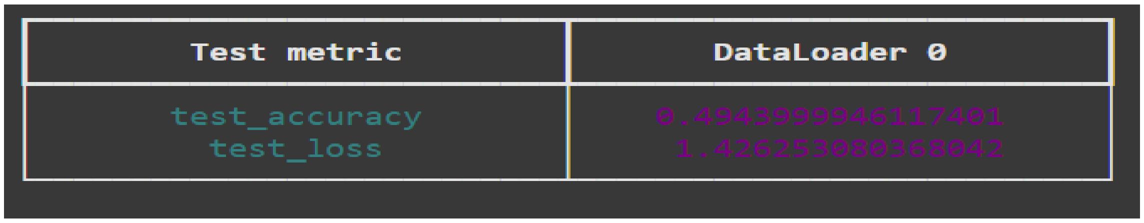
The validation loss decreased continuously throughout training, from above 1.52 to around 1.41. This trend suggests that the model's predictions on validation data became more reliable and confident over time, indicating successful fine-tuning without signs of overfitting.



Run ↑	Smoothed	Value	Step	Relative
● version_2	1.4241	1.4125	2,891	26.69 min

Test Accuracy:

The final test accuracy achieved by the fine-tuned model was 49.43%. This performance is close to the validation accuracy, showing that the model generalized reasonably well to unseen test data. However, it was notably lower than the 72.18% accuracy achieved when training from scratch, as expected due to the limited training of only the classifier layer.



```
┃        Test metric        ┃       DataLoader 0        ┃
│       test_accuracy       │    0.49439999461174
│         test_loss         │     1.426253080368042     │
```

Summary

In this report, I explored the performance of several convolutional neural network (CNN) architectures on the CIFAR-10 dataset. I began with a basic CNN model as a baseline, followed by a custom All-CNN architecture designed without pooling layers, relying instead on strided convolutions. To enhance performance and reduce overfitting, I introduced regularization techniques such as dropout and batch normalization. Finally, I implemented transfer learning using a previously trained All-CNN model from the Imagenette dataset. I fine-tuned this model on CIFAR-10 by freezing the convolutional layers and training only the final classifier layer. Each model was evaluated using test accuracy and test loss. Among all models, the best performance came from training All-CNN from scratch on CIFAR-10, achieving a test accuracy of 72.18%. While fine-tuning with transfer learning achieved a lower accuracy of 49.43%, it still demonstrated the potential of leveraging pretrained models, especially in environments with limited training time and compute resources. Overall, this report highlights the effectiveness of model architecture design and training strategies like regularization and transfer learning in improving CNN performance on image classification tasks.