Daniel-Cristian-Marian Țăpuși

Neural Networks - HW1 report

Task 1:

In implementing the forward pass of the MyConvStub class, I handled the core aspects of convolutional operations, ensuring that parameters such as stride, dilation, and grouping were accurately integrated.

I started by computing the output dimensions, which involved the height and width of the output tensor using these formulas:

$$\text{Out\_h} = \left[\frac{height - (\text{ker}nel_{height} - 1) \times dilation - 1}{stride} + 1\right]$$

$$\text{Out\_w} = \left[\frac{width - (\text{ker}nel_{width} - 1) \times dilation - 1}{stride} + 1\right]$$

The next step was to calculate the number of input channels and filters per group to confirm that each group processes a different subset of input channels.

Before starting to loop over the groups, I extracted from the input tensor the patches involved in the convolution operation, considering both stride and dilation.

Then, for each group, I extracted the corresponding subset of weights based on the group index and performed the matrix multiplication with the extracted patches. The result was reshaped to match the expected dimensions of the output. Also, the bias term was firstly reshaped and then added to the final result of the convolution.

In implementing the forward pass of the MyFilterStub, I implemented the channel-wise application of the blur filter, meaning that the same filter was applied indpendently to each input channel.

Similar to MyConvStub, I started by computing the output dimensions:

$$\text{Out\_h} = \text{height} - \text{kernel\_h} + 1$$

$$\text{Out\_w} = \text{width} - \text{kernel\_w} + 1$$

Then I traversed the matrix and extracted a patch from all channels of the input tensor, that matches the size of the blur filter and performed the element wise multiplication between the extracted patch and the blur filter.

```
danyez87@fedora:~/Master AI/NN/HW1$ /bin/python3.10 "/home/danyez87/Master AI/NN/HW1/test_conv.py"
............
----------------------------------------------------------------------
Ran 12 tests in 15.573s

OK
```

Task 2:

I designed a CNN named ImagenetteModel, structured to contain 3 convolutional layers followed by 2 fully connected layers, which respects the constraint of being at most 5 layers deep. The architecture is as follows:

- **The first convolutional layer** is made of a 7x7 kernel with a padding of 3 and 32 output channels, followed by batch normalization (which is applied in some of the experiments), a ReLU activation function, a MaxPooling layer with a 2x2 window and dropout layer of 0.3, that is also optional.
- **The second convolutional layer** uses a 5x5 kernel with a padding of 2 and 64 output channels, followed by a similar optional batch normalization, ReLU activation function, MaxPooling and an optional dropout layer.
- **The third convolutional layer** consists of a 3x3 kernel with a padding of 1 and 128 output channels, continued by the same sequence, as the layers from above.
- **The first fully connected layer** connects the output from the convolutional layers to 512 neurons, followed by an optional batch normalization, ReLU activation function and an optional dropout layer.
- **The second fully connected layer** maps the 512 neurons to the 10 output classes corresponding to the Imagenette categories.

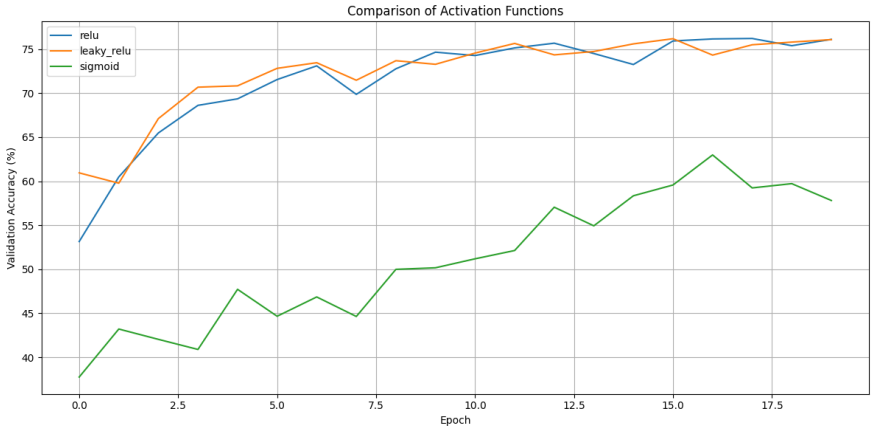
During the development of the ImagenetteModel, careful consideration was given to the configuration of convolutional layers to enhance the model's performance. I started by experimenting with varying numbers of filters across the convolutional layers, with smaller ones such as 8, 16, and 32. However, as the network deepened, increasing the number of filters to 16, 32, and 64 enabled the model to learn more complex and abstract representations and so, obtaining an improved accuracy. Starting with 32 filters in the first layer and doubling them in each convolutional layer that comes after allows the network to learn increasingly complex and abstract features as the depth increases.

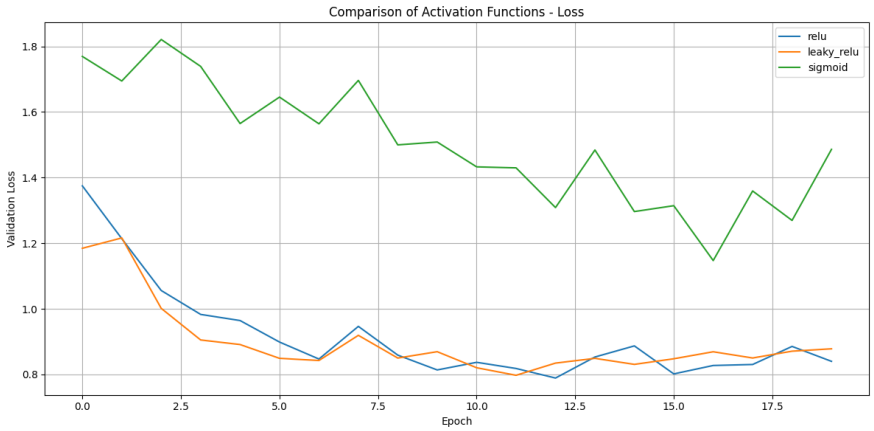
The kernel sizes are larger in the first convolutional layer and decrease in each convolutional layer that follows, to capture broad features, while the deeper layers focus on fine-grained features essential for distinguishing between classes.

The padding sizes are applied corresponding to the kernel sizes to ensure that the spatial dimensions of the feature maps are preserved across layers.

A critical aspect of the model design was selecting the most effective non-linear activation function. I evaluated the impact on both training and test performance that different models with the 3 activation functions (ReLU, Leaky ReLU and Sigmoid) and no other parameter modified would achieve. The experiments showed that ReLU slightly outperformed Leaky ReLU in terms of both convergence speed and final accuracy. While Leaky ReLU's small negative gradient (α=0.01) was designed to prevent the "dying ReLU" problem where neurons can become permanently inactive, this advantage did not translate into notably better performance in our specific case. The sigmoid activation function showed significantly worse results compared to the other two, because of its tendency to saturate at both extremes of its input range, which leads to vanishing gradients. Additionally, its non-zero centered output made the network more prone to saturation issues during training.

The marginal superiority of ReLU over Leaky ReLU, combined with its simpler computational nature, made it the preferred choice for our final model design.



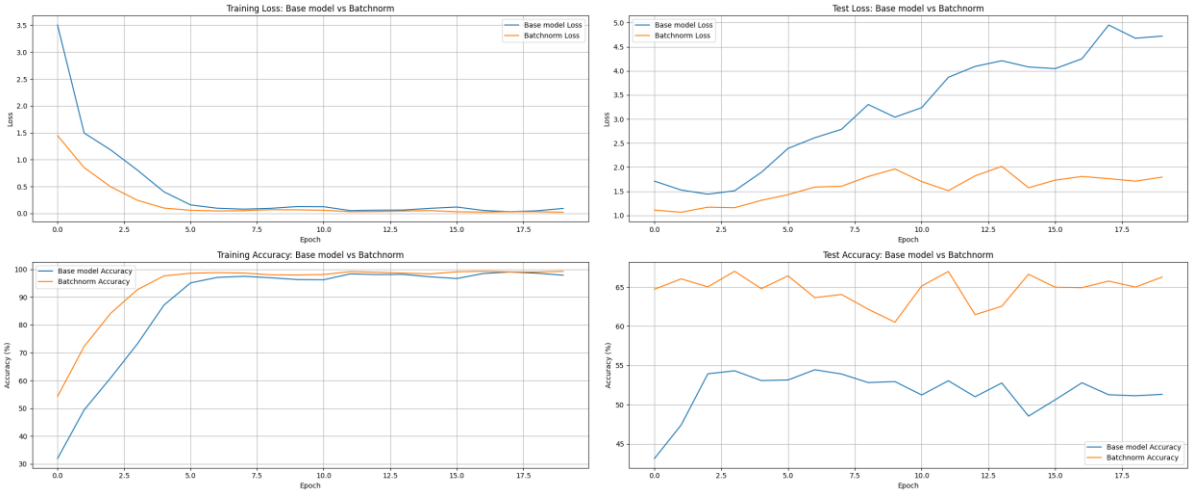


Another critical design decision was the inclusion of Batch Normalization and Dropout layers. The model architecture allows for optional BatchNorm after each convolutional and the first fully connected layer, as well as Dropout layers with a rate of 0.3.

Experiments:

The effect of batch normalization:

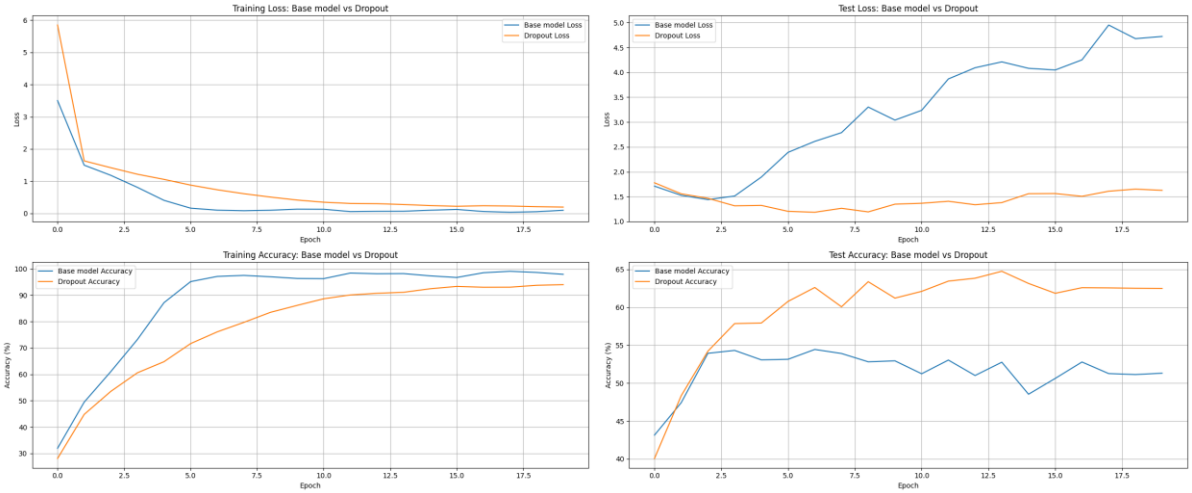
Introducing batch normalization after each convolutional and fully connected layer significantly improved both training and test accuracies. The base model achieved a test accuracy of 51.31%, whereas the batch-normalized model attained a test accuracy of 66.27%. The effectiveness of batch normalization can be attributed to several key mechanisms. First, it stabilizes the training process by normalizing the inputs to each layer, reducing the internal covariate shift that can hinder learning. This normalization allows the use of higher learning rates without the risk of divergence, accelerating the training process.



The effect of dropout:

Incorporating dropout layers (with a rate of 0.3) after convolutional and fully connected layers proved to be an effective strategy against overfitting. The experimental results showed a significant improvement, with the dropout-enhanced model achieving a test accuracy of 62.50% compared to the base model's 51.31%. During training, dropout randomly deactivates 30% of neurons, forcing the network to develop more robust and redundant feature representations. This prevents the model from becoming overly dependent on specific neurons or features, leading to better generalization.

Interestingly, while the dropout model showed slightly lower training accuracy compared to the batch-normalized model, its improved test performance indicates successful mitigation of overfitting.

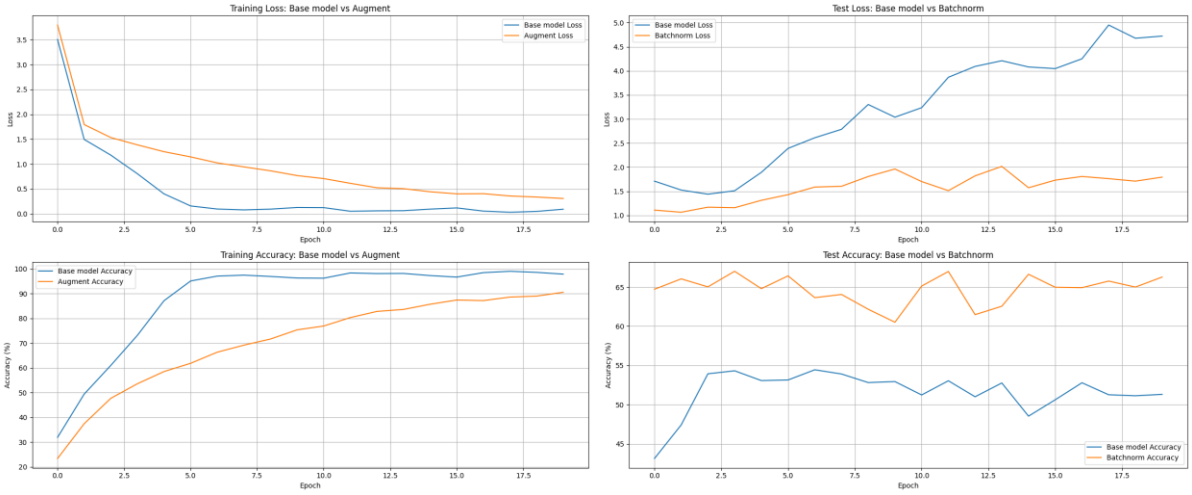


The effect of data augmentation:

The implementation of data augmentation techniques proved to be a powerful strategy for improving model generalization, resulting in a significant boost in test accuracy to 64.03%. This enhancement was achieved using these 3 key components:

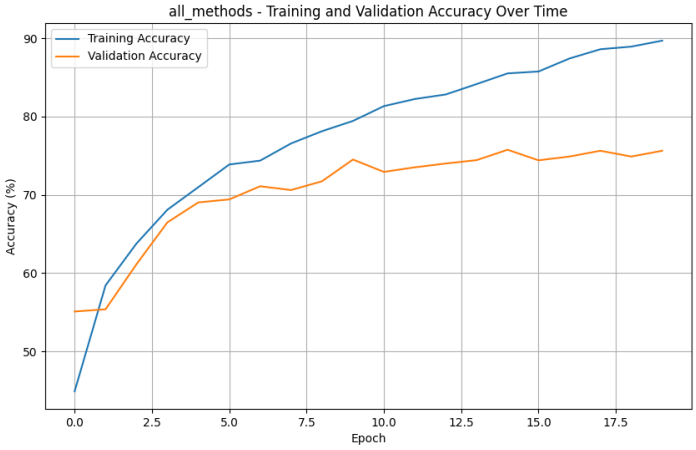
- **Random horizontal flips**: By randomly mirroring images horizontally, the model learned to recognize objects regardless of their horizontal orientation.
- **Color jittering**: The implementation included random adjustments to brightness, contrast, and saturation (with a factor of 0.2), helping the model become invariant to different lighting conditions and color variations that can be found in real-world scenarios.
- **Random rotations:** Small random rotations within ±10 degrees were applied, enabling the model to better handle slightly tilted images and varying orientations.

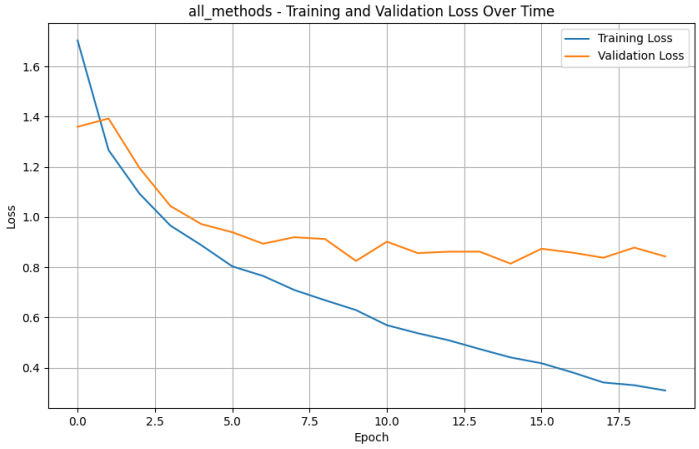
The substantial improvement in test accuracy proves that this augmentation strategy minimized overfitting by exposing the model to a wider range of input variations, thereby enhancing its ability to perform well on unseen test data.

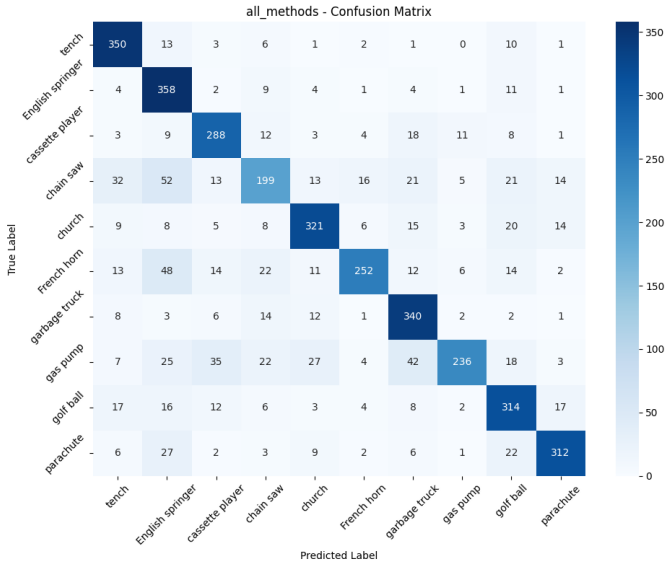


Combined Effects of Normalization, Dropout, and Data Augmentation:

The combination of all three enhancement techniques - batch normalization, dropout, and data augmentation - proved to be significantly more powerful than applying each method individually. This comprehensive approach led to a remarkable test accuracy of 75.67%, representing a dramatic improvement of over 24 percentage points compared to the base model's 51.31%. This superior performance demonstrates how each technique addressed different aspects of the learning process. While batch normalization enabled faster and more stable training, dropout ensured the network did not rely too heavily on specific features. Meanwhile, data augmentation provided the model with a richer and more diverse set of training examples. The combination of all three techniques created a robust learning environment where the model could develop stable, generalized features while maintaining strong regularization.


all_methods - Training and Validation Accuracy Over Time
Training Accuracy
Validation Accuracy
Accuracy (%)
Epoch


all_methods - Training and Validation Loss Over Time
Training Loss
Validation Loss
Loss
Epoch


all_methods - Confusion Matrix
True Label
Predicted Label
tench
English springer
cassette player
chain saw
church
French horn
garbage truck
gas pump
golf ball
parachute

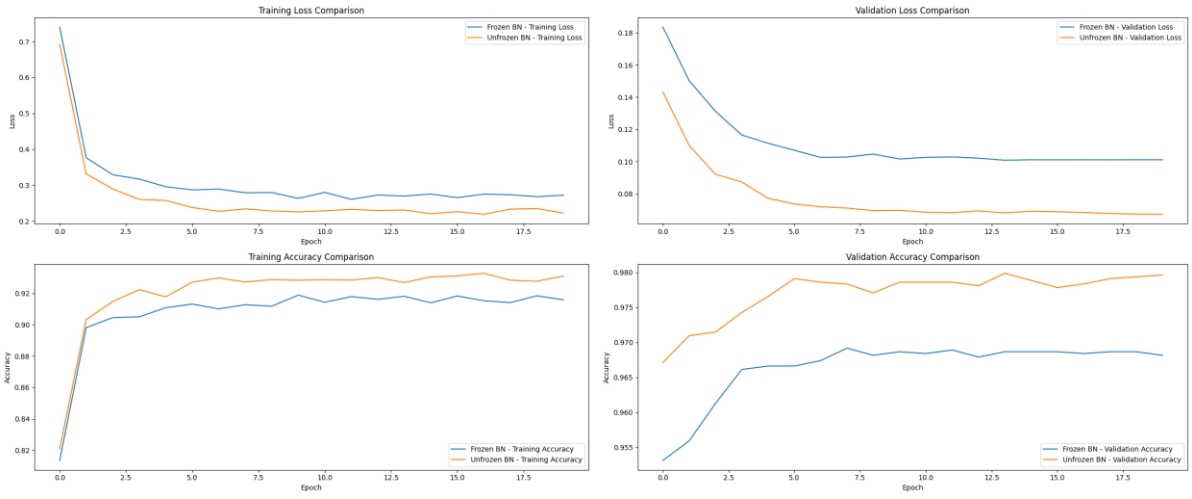
Task 3:

I began by importing the ResNet-18 model from PyTorch's torchvision library, which is pre-trained on the ImageNet dataset. To tailor ResNet-18 for the Imagenette dataset, I replaced the final fully connected (FC) layer to match the number of classes in Imagenette (10 classes). This modification allows the network to output logits corresponding to the specific categories within the dataset.

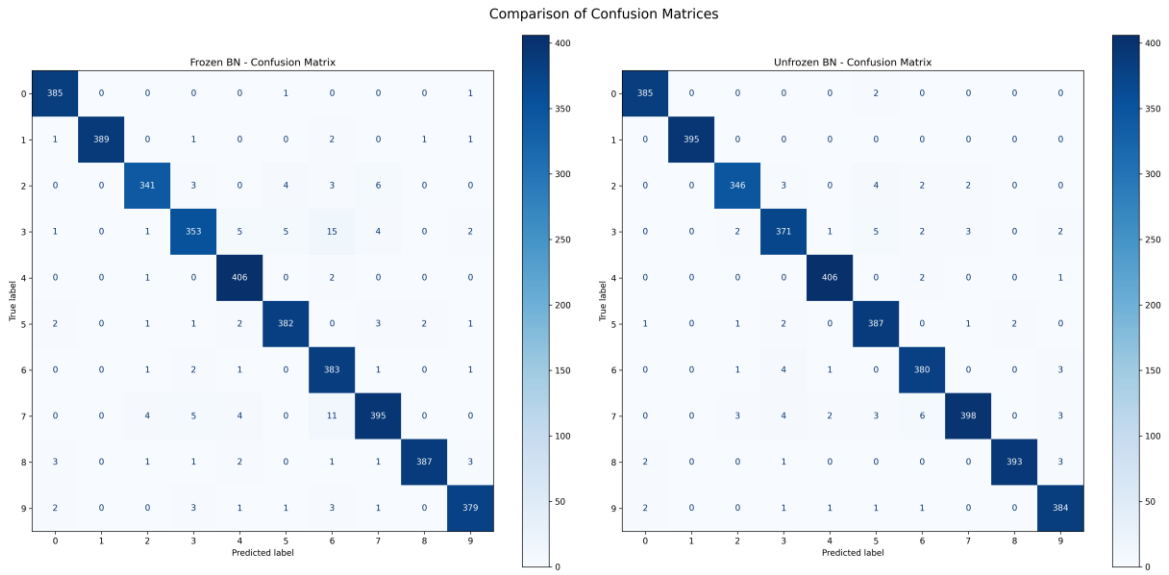
Initially, all parameters of ResNet-18 were frozen to preserve the pre-trained weights. However, recognizing that BatchNormalization layers maintain running mean and variance statistics that are sensitive to the specific dataset, I performed the unfreezing of the layers, so the model can adapt its normalization statistics to better fit the Imagenette dataset.

Results:

The frozen BN layers configuration displayed a steady decrease in training loss, reaching a low point by the end of training. Similarly, the unfrozen BN layers configuration showed a similar trend, but achieved a slightly lower test loss, indicating improved convergence and better generalization.



The accuracy plots revealed that the model with unfrozen BN layers consistently outperformed the baseline across all epochs. Starting from an initial test accuracy of approximately 43.72%, the unfrozen BN configuration achieved a higher test accuracy of **97.96%** by the end of training.



The model with frozen BN layers achieved a test accuracy of **96.82%**, displaying strong performance with some misclassifications in similar classes, while the model with unfrozen BN layers demonstrated a slightly higher test accuracy of **97.96%.**

Comparison with Task 2:

Comparing the fine-tuned ResNet-18 model to the simple CNN model developed in Task 2 reveals a substantial performance boost. While the best performing simple model from Task 2 achieved a test accuracy of **75.67%** with all training enhancements, the fine-tuned ResNet-18 model achieved a test accuracy of **97.96%**. The superior performance of the unfrozen BN configuration can be attributed to the layers' ability to adapt their normalization statistics to the specific characteristics of the Imagenette dataset, resulting in more optimal feature distributions, but also thanks to the effectiveness of transfer learning and the use of pre-trained models, that can achieve a superior performance with fewer computational resources and training time.

Impact of unfreezing the BN layers:

We can conclude that the improvement between the two ResNet-18 models came from the ability of the BN layers to adapt their running statistics to the specific characteristics of the Imagenette dataset. This happend thanks to this freedom given to layers to update, the model better normalized feature distributions, which led to more accurate classifications.