1 Base Model

The other two models are built off the base model.

Architecture The architecture is the same as VGG 16 configuration "D" with batch normalization and transfer learns from it. Batch normalization occurs after each convolution layer before the activation function. The entire model is trained, not just the classification layers. The last layer is replaced to accept the same number of inputs (4096 features) but outputs 100 features.

I decided on VGG 16 (instead of 19) because the resolution of the images from CIFAR-100 are 32 by 32 pixels, providing a total of 1024 features. The pretrained models provided by PyTorch are trained on images from ImageNet, which are 224 by 224 pixels, providing a total of 50,176 features. The pretrained models are trained to capture many more features, so have many more parameters. To prevent overfitting, I settled on a smaller model.

Hyperparamters The learning rate starts at 0.1 but is multiplied by 0.2 after every 110 epochs. The epoch step size was determined by observing the epochs that did not have significant improvements in development accuracy ($> 1\%$). The multiplication factor was selected after trying values from various magnitudes.

Momentum is set to 0.9; weight decay parameter is set to 0.00001; the number of epochs is set to 200; and the batch size is 128.

Augmentations From initial prototyping, I realized that augmenting the training data greatly reduced overfitting. This is especially true for the rotation augmentation, which improved accuracy by over 4% as seen in the abalation study table. Consequently, I spent a great deal of time selecting and testing transformation provided by PyTorch.

First, I prioritize transformations that do not distort the intrinsic qualities of an object like shape, texture, and position. I employed as many of these transformation as I could find. The following are transformations I used:

- Random Color Jitter: randomly change the contrast and brightness of an image.
- Random Crop: randomly crop the image (though CNN have built in translation invariance due to max pool, I hoped this transformation will provide a greater translation invariance)
- Random Horizontal flip: randomly flip image along the vertical axis in the center (after examining sample images, I realized that most, if not all, the images can be horizontally flipped without affecting the content of the image)
- Random Rotation: to preserve the position of the image, only rotations of at most 15 degrees (in either direction) are permitted.

One transformation that I used that may distort the intrinsic qualities of an object is random erasing, which erases a rectangular shape (of random size) from the image. I believe this transformation will reduce overfitting by preventing the model from over depending on a specific feature of an object.

One transformation that distorts an object that I stopped using is random perspective, which performs an algorithm with interpolation to change the perspective of the image. This transformation reduced accuracy by around 1% as seen in the abalation study table. I believe this transformation has a negative effect because the input images are relatively small, but the change in perspective removed some pixel values. Further, from my qualitative evaluation, the transformed images are hard to recognize.

Training Though the epochs is set to 200, I used an early stopping mechanism so I don't waste time with unnecessary training. The early stopping has a patience of 15 epochs: when the current best accuracy hasn't improved in over 15 epochs, the training stops. The training uses cross entropy loss with stochastic gradient descent.

Testing During testing, I did not use any transformations on the input image; however, I used a five crop method, which pads the image with 4 pixels, takes a crop at the center and four corners, pass them through the model, average the predictions, then generate a classification. This method improved the accuracy by approximately 0.1%.

2 Derived Models

Model 2 To see the effects of batch normalization, model 2 is VGG 16 without batch normalization. Since batch normalization allows a single instance to be seen in conjunction with other samples in the mini-batch (acting as a regularizer) and stabilizes the gradients, I expected the accuracy to decrease in this model. All other configurations are the same as the base model expect for the learning rate, which was reduced to 0.01.

Model 3 In model 3, I used the same configuration as the base model, but removed dropout from the classification layers. Since dropout helps reduce overfitting by reducing a model's dependency on any one given neuron, I expected the accuracy to be lower than the base model's.

Model 4 Model 4 is purely for the purpose of the abalation study. This model introduces a perspective transformation on the input images using PyTorch's RandomPerspective method. All other configurations are the same as the base model.

Model 5 Model 5 is purely for the purpose of the abalation study. This model removes the slight rotation transformation on input images. All other configurations are the same as the base model.

3 Abalation Study Tables

Model Name	Description	Time (min)	Best Dev Acc. (%)
Base Model	The base model	130.4	71.4
Model 2	No BN	26.9	66.10
Model 3	No Dropout	108.3	68.0
Model 4	Perspective Change	81.6	70.0
Model 5	No Rotation	52.8	67.3

Notes Batch normalization affected the accuracy the most among the other manipulations for this task. One thing I notice during training is that model 2's accuracy increased the fastest among the other models during the first few epochs. Further, model 2 performed worse than the other models but was the fastest (around 4x faster than model 3 and 5x faster than the base model).

4 Best Model Plots

The following are plots for the base model, which is the best performing model.