Part 1

Kaggle username: gargsid1711

Best Accuracy: 48.4%

Neural Net Architecture

Layer No.	Layer Type	Kernel Size	Input|Output dimensions	Input|Output channels (for convolution layers)
1	Conv2d-1	3	32|32	3|16
2	BatchNorm2d	-	32|32	-
3	ReLU	-	32|32	-
4	Maxpool2d	2	32|16	-
5	Conv2d-2	3	16|16	16|32
6	BatchNorm2d	-	16|16	-
7	ReLU	-	16|16	-
8	Maxpool2d	2	16|8	-
9	Conv2d-3	3	8|8	-
10	BatchNorm2d	-	8|8	32|64
11	ReLU	-	8|8	-
12	Maxpool2d	2	8|4	-
13	Conv2d-4	3	4|4	64|128
14	BatchNorm2d	-	4|4	-
15	ReLU	-	4|4	-
16	Linear	-	(128*4*4)|1024	-
17	ReLU	-	1024|1024	-
18	Linear	-	1024|512	-
19	ReLU	-	512|512	-
20	Linear	-	512|256	-
21	ReLU	-	256|256	-
22	Linear	-	256|100	-

Factors that helped improve model performance:

Convolution Layers:

Increasing the number of convolution layers improved the validation accuracy. The convolution layers were designed with increasing sizes of output channels in factors of 2. Each convolution layer was followed by a ReLU activation function.

Linear Layers:

Increasing the number of linear layers after the last convolution layer led to a significant increase in the validation accuracy achieved. The linear layers were designed with decreasing output sizes in factors of 2 and followed by ReLU activation functions except the last linear layer.

Normalization Layers:

For the normalization of the convolution layers using the BatchNorm2d function, the model performed better when the normalization was done before the ReLU function was applied to the convolution outputs.

Normalizing after linear layers using BatchNorm1d improved the accuracy to 49%. Here, performing the normalization before the ReLU was applied also performed better.

Pooling was done after each ReLU layer except the last to reduce the computation cost of the network and prevent overfitting of the data.

Tuning Hyperparameters:

Finally increasing the learning rate from 0.005 to 0.01 and increase in epochs to 20 led to another an improvement in the performance.

Trying to add stacks of convolution layers with the same input and output features resulted in the accuracy dropping by 3%. So layered layers were not used in the architecture.

Transformations Used:

1. RandomHorizontalFlip()
2. RandomRotation(10)
3. Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))

Applying the transforms before making any changes to the net improved the model's performance by 6%.

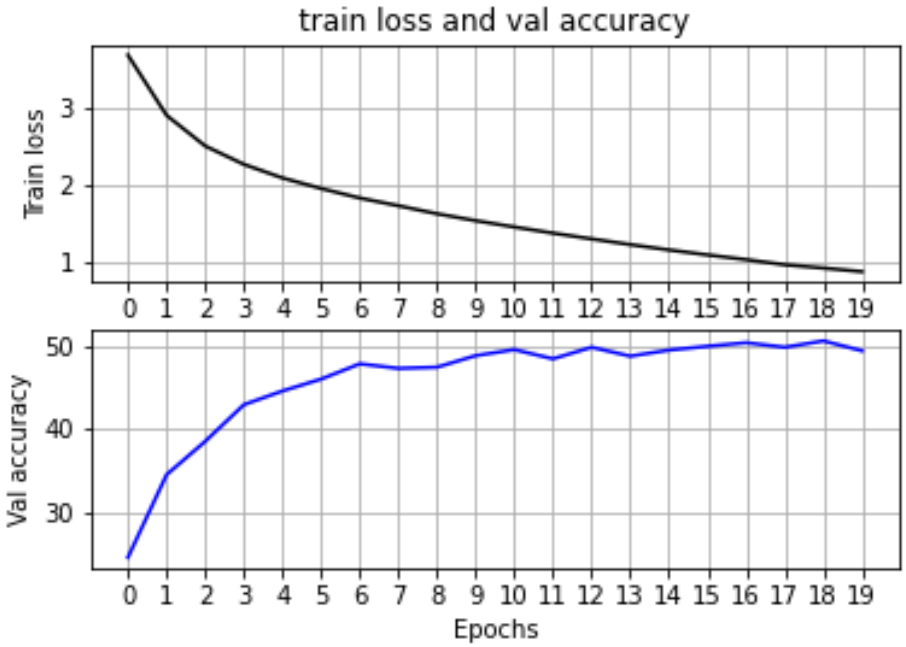

Figure 1: Final plot for training loss and validation accuracy

Ablation Study

The hyperparameters at the start of the study are as follows:

EPOCHS = 15, LR = 0.005

Features in model	Loss	Validation accuracy
BaseNet	2.879	21%
+ Data Augmentation and Normalization	2.847	27%
+ Convolution Layers + Pooling	1.815	40%
+ Linear Layers + Normalization Layers (2d on conv, 1d on linear)	1.488	45%
+ tuning hyperparameters (20 epochs and lr=0.01)	1.252	50%

Part 2

ResNet Mode	Train Accuracy	Val Accuracy
Fixed-feature extractor	72.3%	38.3%
Fine-tuning the whole network	99.4%	61%

Hyperparameters:

Epochs = 20

At the learning rate of 0.01, the model could still improve its accuracy if was given more than 15 epochs

Learning Rate = 0.001

At the lower learning rate of 0.0001, the model improved in accuracy very slowly and did not go past 30% after 20 epochs. The LR was gradually increased from 0.0001 to 0.001 and the performance improved. Increasing beyond this would cause the model to overfit the data and perform poorly on the Kaggle test set.

Batch Size = 16

Increasing the batch size from 8 to 16 made the model more efficient as it processed more images at a time and improved the performance of the model.

Transformations:

Transforms Used	Training accuracy	Val Accuracy
None	99.75%	56.7%
+ CenterCrop(224)	99.75%	56.1%
+ RandomHorizontalFlip()	99.83%	60.83%
+ RandomRotation(15)	99.25%	59.17%

(These values were recorded while fine-tuning the entire network)

I noticed that without any augmentations, the model converges very quickly to a high training accuracy while still having a low validation accuracy. This is because the model overfitted on the training set. This also led to the Kaggle test set accuracy to be very low, ranging from only 10-20%. Adding the augmentations changes the training set and prevents overfitting. Even though adding the transformations didn't significantly improve the validation accuracy, the test accuracy on Kaggle improved significantly from around 20% to 40% and up to 54% after all the augmentations were applied.