

Resnet Neural Network Report

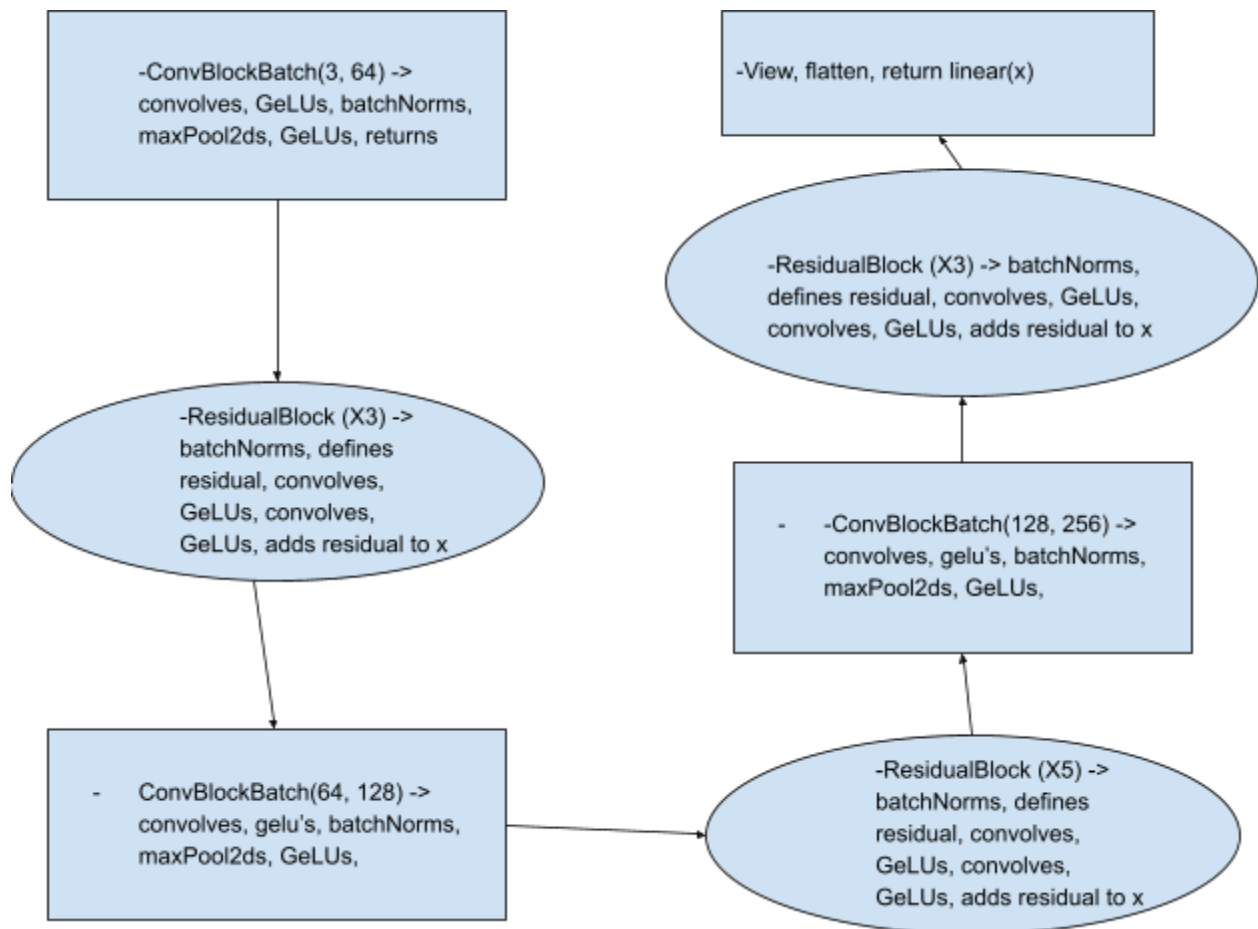
Thomas Knickerbocker

Dataset used: CFAIR-10 (via PyTorch)

Model Architecture, Parameters, & Notes:

My model uses 3 epochs, a batch size of 500, a step size of .0009, GeLU as the loss function, linear as the output function, and a general structure of the following:

Architecture:

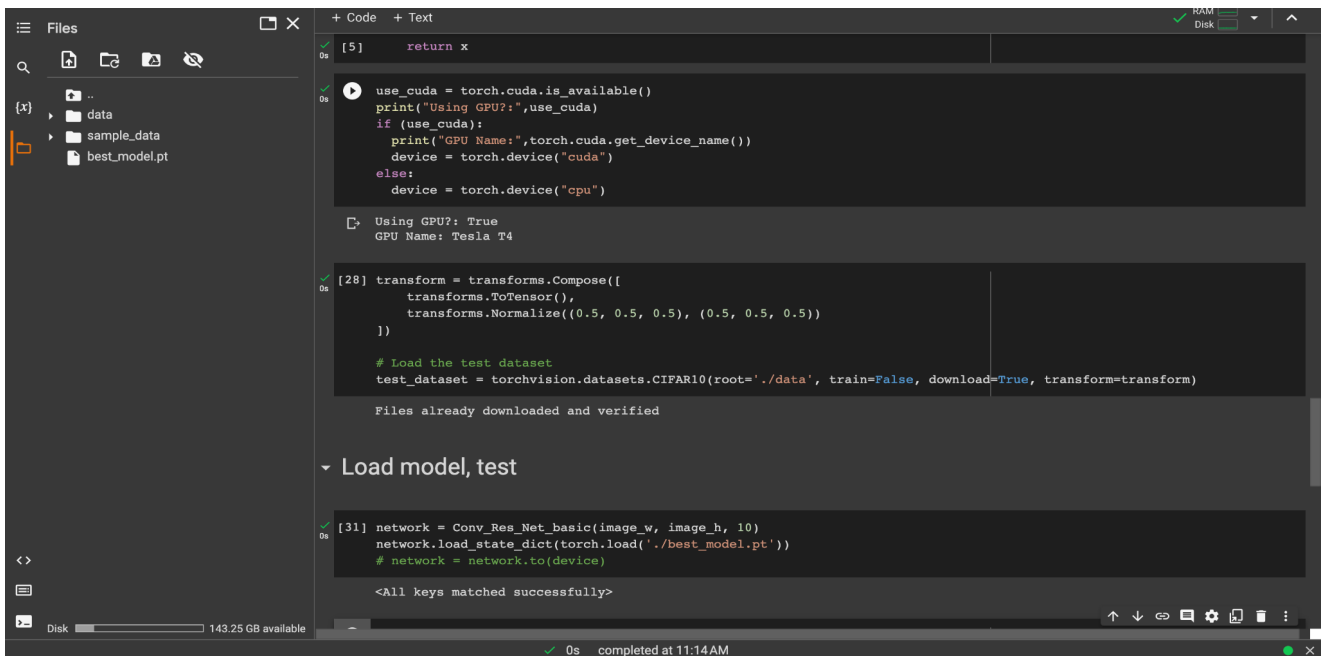


Training data was augmented to be transformed, shifted, rotated, cropped, normalized. This was done to give the model more noisy data to train on, as well as to give it more data overall.

Testing data was only normalized across RGB values, because adding noise to testing data will reduce our model's accuracy on said data. I was never able to increase my accuracy with dropout, surprisingly, and adding more than 3 blocks reduced my accuracy as well. Some of the best decisions as it pertains to increasing the accuracy of my model were implemented batch normalization (helping us further normalize our data), residuals for forward passes (which help

alleviate the zero gradient problem), and setting num_kernels to high numbers (which helps us retain some information following the loss that occurs from maxpools). I used GeLU because it has less discontinuity as an activation function when compared to ReLU, and when tested, it improved my results. Channel sizes between blocks of 128, 256, and 512 worked the best (although having 64, 128, & 256 as sizes were fairly close behind). Going beyond these meant parameter numbers far into the millions, and training times of more than five minutes. I chose the max batch size (500) as mine because colab was able to handle it, memory-wise, and there is no point in choosing anything smaller, given the computing power available to me for this assignment.

Loaded into the second submitted file, loadModel:



The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files 'data', 'sample_data', and 'best_model.pt'. The code editor contains the following code:

```
[5] return x

use_cuda = torch.cuda.is_available()
print("Using GPU?:", use_cuda)
if (use_cuda):
    print("GPU Name:", torch.cuda.get_device_name())
    device = torch.device("cuda")
else:
    device = torch.device("cpu")

Using GPU?: True
GPU Name: Tesla T4

[28] transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Load the test dataset
test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

Files already downloaded and verified

Load model, test

[31] network = Conv_Res_Net_basic(image_w, image_h, 10)
network.load_state_dict(torch.load('./best_model.pt'))
# network = network.to(device)

<All keys matched successfully>
```

The bottom status bar indicates 'Disk 143.25 GB available' and 'completed at 11:14 AM'.

Accuracy & confusion matrix for loaded-in model on testing data:

Files

(x)

data

sample_data

best_model.pt

+ Code + Text

```
network.eval()
allPreds, allTargs = [], []

# Predict
with torch.no_grad():
    for img, label in test_dataset:
        outputs = network(img.unsqueeze(0))
        _, predicted = torch.max(outputs.data, 1)
        allPreds.append(predicted.item())
        allTargs.append(label)

# Get accuracy, display confusion matrix
accuracy = sum([1 if allPreds[i] == allTargs[i] else 0 for i in range(len(allPreds))]) / len(allPreds)
print(f"Accuracy: {accuracy}")

cm = confusion_matrix(allTargs, allPreds)
print(f"Confusion matrix:\n{cm}")
```

Accuracy: 0.7891
Confusion matrix:
[[779 11 66 22 30 1 6 12 57 16]
[13 900 3 6 4 3 1 3 21 46]
[48 3 752 48 59 34 27 22 5 2]
[8 7 72 642 77 88 68 21 10 7]
[13 2 70 52 777 13 21 42 9 1]
[8 3 50 184 47 644 14 43 4 3]
[3 5 57 37 22 9 857 4 4 2]
[7 0 21 45 55 26 5 835 3 3]
[43 14 11 7 6 0 4 6 901 8]
[45 76 11 13 7 2 1 11 30 804]]

[35] print(classes)

('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

Disk 143.25 GB available

0s completed at 11:14AM