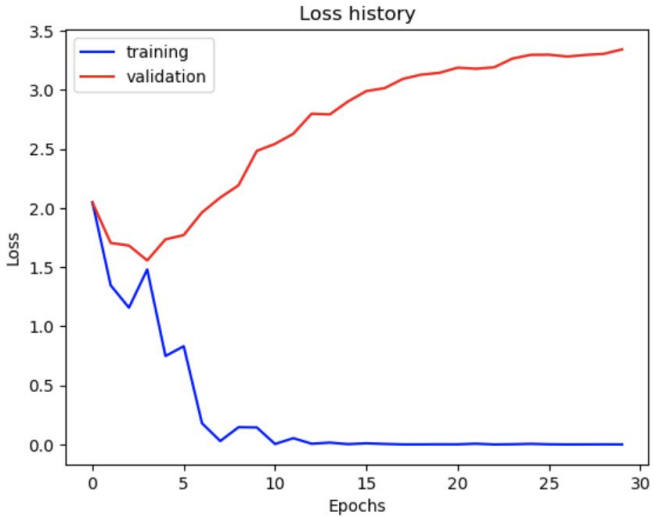
CS 6320 Project 6

Disha Kunjadia

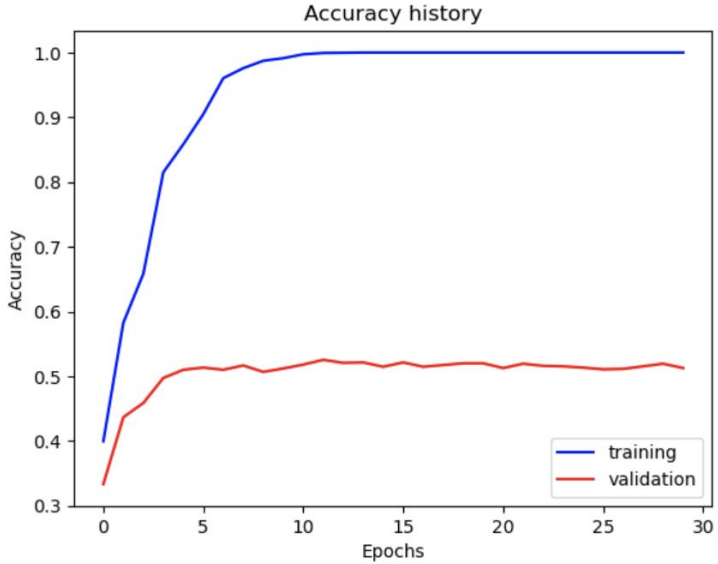
u1464153

Part 1: Your Training History Plots

<Loss plot here>



<Accuracy plot here>



Train Accuracy = 1.0
Validation Accuracy = 0.51266

Part 1: Experiment: play around with some of the parameters in nn.Conv2d and nn.Linear, and report the effects for: 1. kernel size; 2. stride size; 3. dim of nn.Linear. Provide observations for training time and performance, and why do you see that?

It took 7 mins to train.

Kernel size: (3, 3)

This kernel is standard for extracting fine-grained spatial features in modern CNNs.Smaller kernel sizes are computationally efficient and work well when combined with multiple layers.

Stride size: (1, 1)

padding=(1, 1))

Stride 1: preserves spatial resolution for feature extraction and ensures no information is skipped during feature extraction.

Dim of nn.Linear:

First layer: `(8192, 128)`
Second layer: `Linear(128, 15)` (15 output classes).

The large input size of 8192 comes from flattening the feature maps from the convolutional layers.

Part 2: Screenshot of your get_data_augmentation_transforms() [code and image example]

```
aug_transforms = transforms.Compose([
      transforms.RandomHorizontalFlip(),
      transforms.RandomRotation(15),
      transforms.Resize(inp_size),
      transforms.ColorJitter(brightness=0.2,contrast=0.2,saturation=0.2, hue=0.1),
      transforms.ToTensor(),
      transforms.Normalize(mean=pixel_mean.tolist(), std=pixel_std.tolist())
  ])
    #raise NotImplementedError('get_data_augmentation_transforms not
implemented')
  return aug_transforms
pixel_mean = np.array([0.485, 0.456, 0.406])
pixel_std = np.array([0.229, 0.224, 0.225])

unnorm = transforms.Normalize(
    mean=(-pixel_mean / pixel_std).tolist(),
    std=(1 / pixel_std).tolist()
)

def tensor_to_image(tensor):
    tensor = unnorm(tensor)
    tensor = torch.clamp(tensor, 0, 1)
    img = tensor.permute(1, 2, 0).cpu().numpy()

    return (img * 255).astype(np.uint8)
```

```
inp_size = (224, 224)

img_path =
"/Users/u1464153/Documents/CV/proj6_6320/data/train/Bedr
oom/image_0040.jpg"
original_img = Image.open(img_path).convert("RGB")
transforms_func =
get_data_augmentation_transforms(inp_size, pixel_mean,
pixel_std)

plt.figure(figsize=(15, 6))
augmented_imgs = [transforms_func(original_img) for _ in
range(5)]
plt.subplot(1, 6, 1)
plt.title("Original")
plt.imshow(original_img)
plt.axis("off")

for i, img in enumerate(augmented_imgs, 2):
    augmented_img = tensor_to_image(img)
    plt.title(f"Augmented {i-1}")
    plt.imshow(augmented_img)
    plt.subplot(1, 6, i)
    plt.axis("off")
plt.show()
```

Augmented 1

Augmented 2

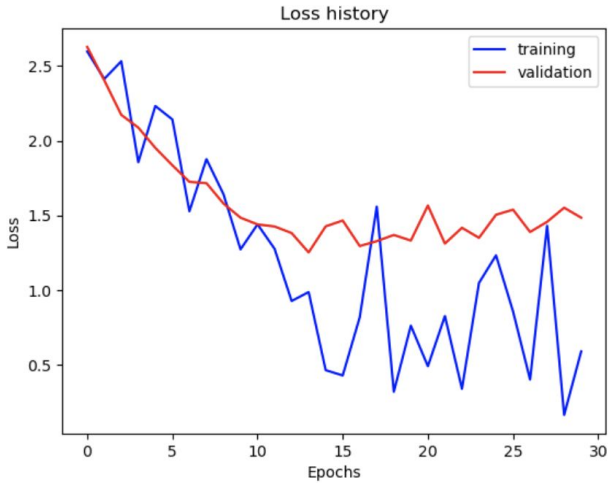
Augmented 3

Augmented 4

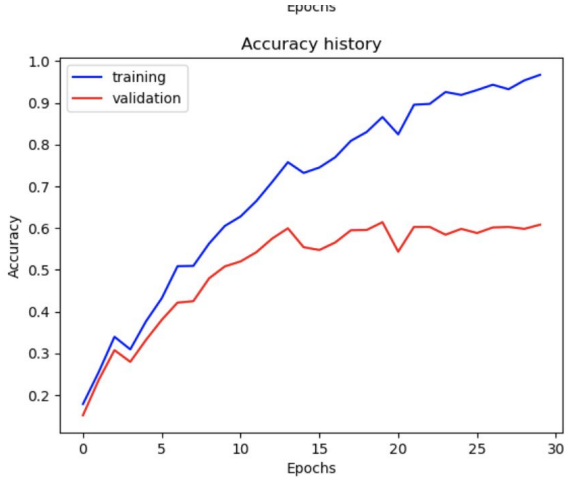
Augmented 5

Part 2: Your Training History Plots

<Loss plot here>



<Accuracy plot here>



Train Accuracy = 0.96716
Validation Accuracy = 0.608

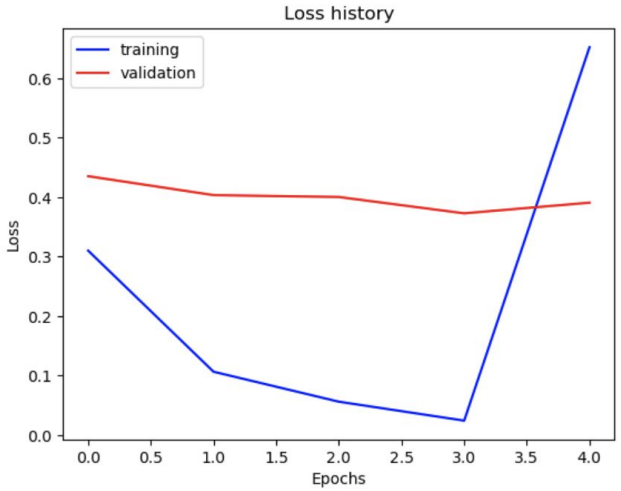
Part 2: Reflection: compare the loss and accuracy for training and testing set, how does the result compare with Part 1? How to interpret this result?

In Part 1, the model achieved a perfect training accuracy of 100%, but the validation accuracy stagnated at around 51%. This shows severe overfitting, where the model memorized the training data but failed to generalize to unseen data. The validation loss either plateaued or increased after a few epochs, further confirming that the model was not learning features that generalized well.

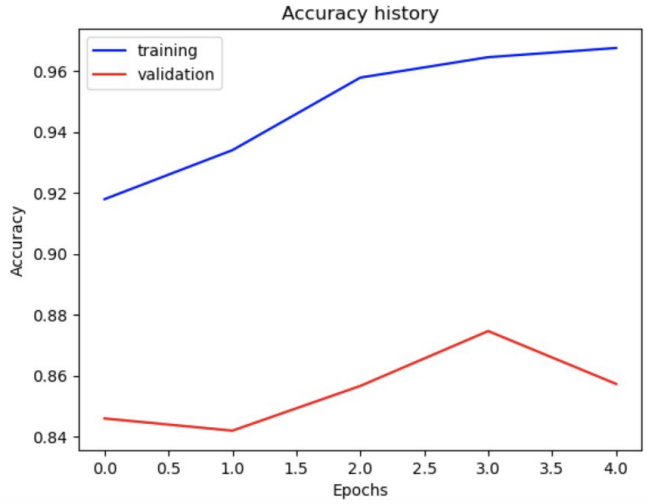
In Part 2, after introducing dropout and data augmentation, the training accuracy dropped slightly to 96.7%, but the validation accuracy improved significantly to 60.8%. The added regularization helped prevent the model from overfitting by reducing reliance on specific neurons and data augmentation. The training loss remained above zero, showing the model was learning more generalizable features instead of just memorizing the data.

Part 3: Your Training History Plots

<Loss plot here>



<Accuracy plot here>



Train Accuracy = 0.967
Validation Accuracy = 0.857

Part 3: Reflection: what does fine-tuning a network mean?

Fine-tuning involves taking a pretrained model and adapting it to a new task by training it further on a different dataset.

So now here we can leverage the general features already learned by the pretrained model, like edges and textures, and adjusts specific layers to specialize in the new task.

The lower layers of the network are frozen since they extract universal features, while the higher layers are fine-tuned to adapt to the specifics of the target dataset.

It saves training time by reusing pretrained features.

Requires less data compared to training from scratch.

Part 3: Reflection: why do we want to “freeze” the conv layers and some of the linear layers in pretrained AlexNet? Why CAN we do this?

Freezing the convolutional layers and some of the linear layers in a pretrained AlexNet learning because it allows us to reuse the powerful features the model has already learned on a large dataset.

These features, such as detecting edges and textures, are general and transferable to many tasks, so retraining them is unnecessary and could lead to overfitting, especially with limited data. Freezing these layers also speeds up training by reducing the number of parameters that need to be updated. We can do this because pretrained weights are robust and have already captured important patterns. That allows us to focus on fine-tuning the higher, task-specific layers for better performance on our target problem.

Conclusion: briefly discuss what you have learned from this project.

This project taught me the essentials of building and training deep learning models for scene recognition using PyTorch. I gained hands-on experience constructing neural networks, understanding key components like convolutional layers, optimizers, and learning schedules, and observing how these affect performance.

Experimenting with different architectures helped me appreciate the trade-offs between model complexity, training speed, and accuracy.

I also learned the importance of tools like data augmentation and pooling for improving generalization in image classification tasks. Overall, this project deepened my understanding of deep learning fundamentals and strengthened my ability to apply them effectively to real-world problems.

Extra Credit

<Discuss what extra credit you did and analyze it. Include images of results as well >