Unstacking Overlaid Fashion MNIST Images

Jordan Stanley, Kenneth Cardona, Maxwell Siebersma

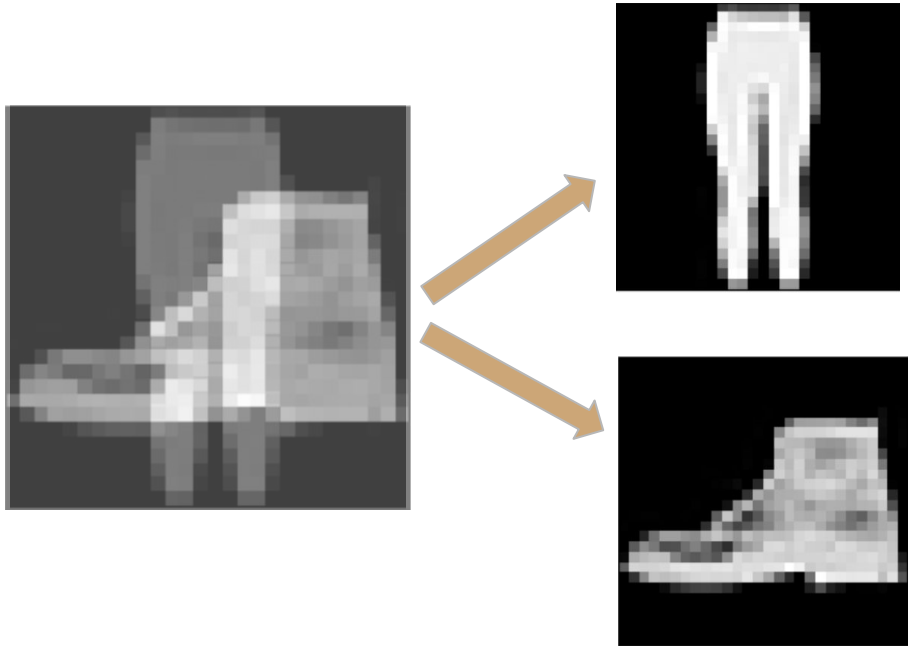
Outline of the Presentation

- Problem Statement & Objectives
- Dataset Description & Preparation
- Model
- Code Outline
- Training & Validation
- Testing
- Challenges
- Takeaways

Problem Statement & Objectives

Problem Statement

- Given a single grayscale image created by overlaying two unknown Fashion MNIST images, can we train a neural network to produce the two original images?



Objectives

- Build a residual convolutional neural network that takes in the overlaid Fashion MNIST images and predicts the original images without seeing them
- If this is successful, can we successfully separate other overlaid grayscale images of the same dimensions? Maybe... professors?

Dataset Description & Preparation

Dataset Description

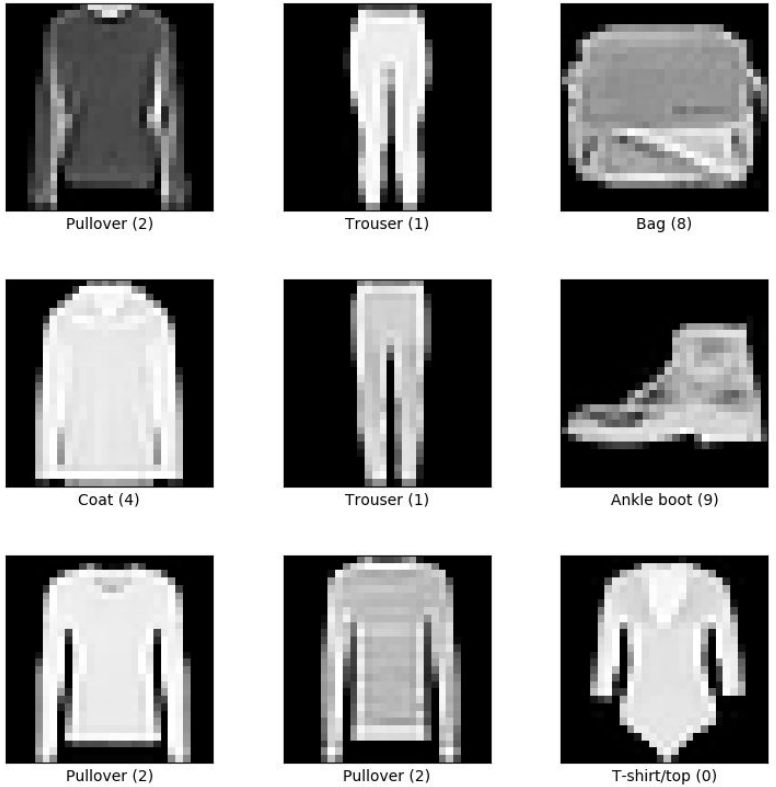
FashionMNIST Dataset: 20,000 Pairs of Images

→ 80% Training

→ 10% Validation

→ 10% Testing

Images randomly selected to be in pairs



Data Preparation

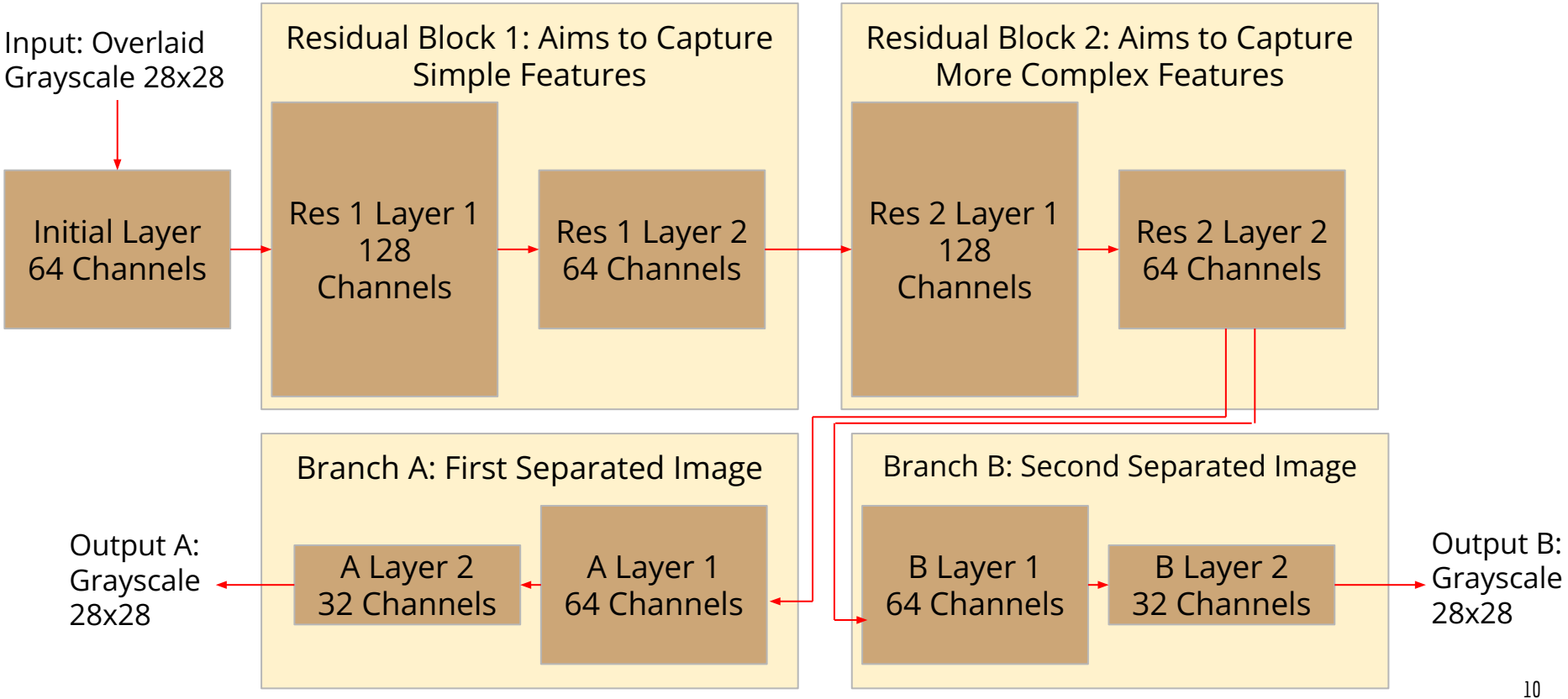
- Downloading the Fashion MNIST dataset is very straightforward (we did it in class)
- To create the overlaid images, we “mixed” the images by averaging the pixel values
  - This forces the neural network to undo a physical mixing process, rather than just trying to separate channels

$$O(x,y) = \frac{1}{2}A(x,y) + \frac{1}{2}B(x,y)$$

Model

CNN Architecture

For all Layers:
Kernel Size 3
1 Pixel Padding



Loss Function

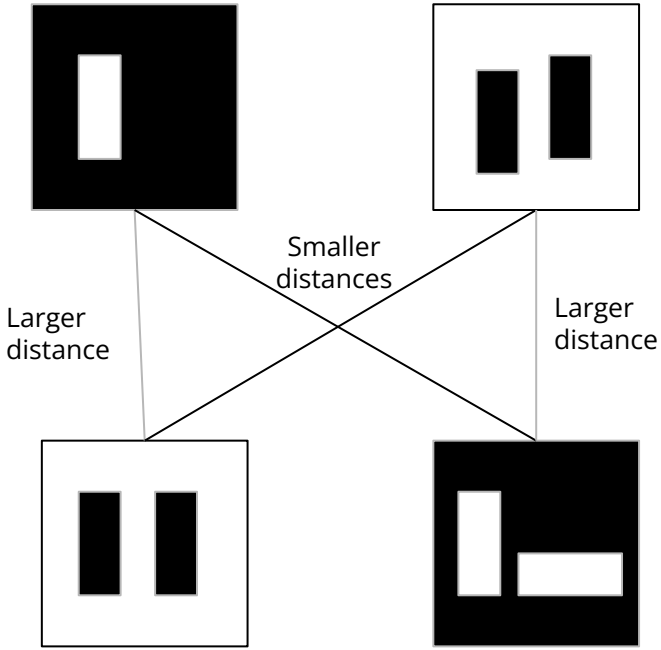
$$d = \sum_{i=1}^{n} |x_i - y_i|$$

Image Distance uses Manhattan Norm (L1)

Model is order-agnostic: does not distinguish between image A and image B

Picks out minimum total distance between all combinations of initial and predicted images

This is simple with two stacked images; (would become computationally expensive with more stacked images and more combinations)



Code Outline

Outline of Code (Dataset)

- Dataset
  - Download the Fashion MNIST dataset
- Overlaying Images
  - Create overlaid dataset, where two random Fashion MNIST images are mixed with an equal weight
- Split Dataset
  - Split the overlaid dataset into training, validation, and testing
- Create Dataloaders
  - Define a batch size of 32, create dataloaders for the training, validation, and testing datasets

Outline of Code (Model)

- CNN Model
  - Takes in overlaid images, returns two predicted images
- Loss Function
  - Used L1 loss, worked the best for predicting the two images
- Device
  - Checked if MPS is available, otherwise use GPU, otherwise used CPU
- Optimizer
  - Used the Adam optimizer
- Show Results
  - Outputs the overlaid image, original images, and predicted images

Outline of Code (Training)

- Training Loop
  - Trained for 40 epochs, did training loop and validation loop sequentially within epochs
  - Reported the training and validation loss every epoch, updated a live plot of the curves to ensure no overfitting
  - Once training was done, used our show results class to see the predictions
- Testing Loop
  - Once training was done, set the model to evaluation mode and ran our testing dataloader through the model
  - Reported the loss value, made sure it's relatively close to the training and validation
  - Also used our show results class to check predictions

Training & Validation

Training Details

- Training
  - Iterates over training dataloader
  - Zeroes grads
  - Forward pass
  - Compute loss
  - Backward pass
  - Update parameters
- Validation
  - Iterates over validation dataloader
  - Forward pass
  - Compute loss

```
# Training and validation loop
for epoch in range(num_epochs):

    # Training
    model.train() # Set model to training mode, enable dropout/batchnorm updates
    metric = d2l.Accumulator(2)  # For accumulating training loss and number of examples
    for overlay, imgA, imgB in train_loader: # Iterate over training data
        overlay, imgA, imgB = overlay.to(device), imgA.to(device), imgB.to(device) # Move data to device
        optimizer.zero_grad() # Zero the gradients
        predA , predB = model(overlay) # Forward pass, get predictions
        loss = separation_loss_l1(predA, predB, imgA, imgB) # Compute loss
        loss.backward() # Backward pass, compute gradients
        optimizer.step() # Update model parameters
        metric.add(loss.item(), overlay.size(0)) # Accumulate loss and number of examples
    avg_train_loss = metric[0] / metric[1] # Average training loss

    # Validation
    model.eval() # Set model to evaluation mode, disable dropout/batchnorm updates
    metric_val = d2l.Accumulator(2) # For accumulating validation loss and number of examples
    with torch.no_grad(): # No gradient calculation during validation
        for overlay, imgA, imgB in val_loader: # Iterate over validation data
            overlay, imgA, imgB = overlay.to(device), imgA.to(device), imgB.to(device) # Move data to device
            predA , predB = model(overlay) # Forward pass, get predictions
            loss = separation_loss_l1(predA, predB, imgA, imgB) # Compute loss
            metric_val.add(loss.item(), overlay.size(0)) # Accumulate loss and number of examples
    avg_val_loss = metric_val[0] / metric_val[1] # Average validation loss
```

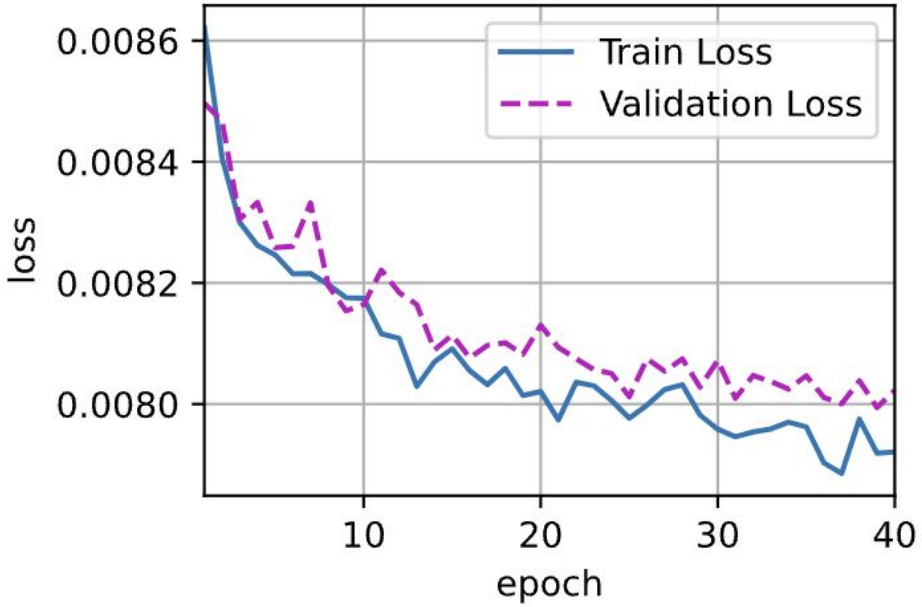
Loss as Function of Epoch and Generalization Error

Asymptotic decay of loss as function of epoch

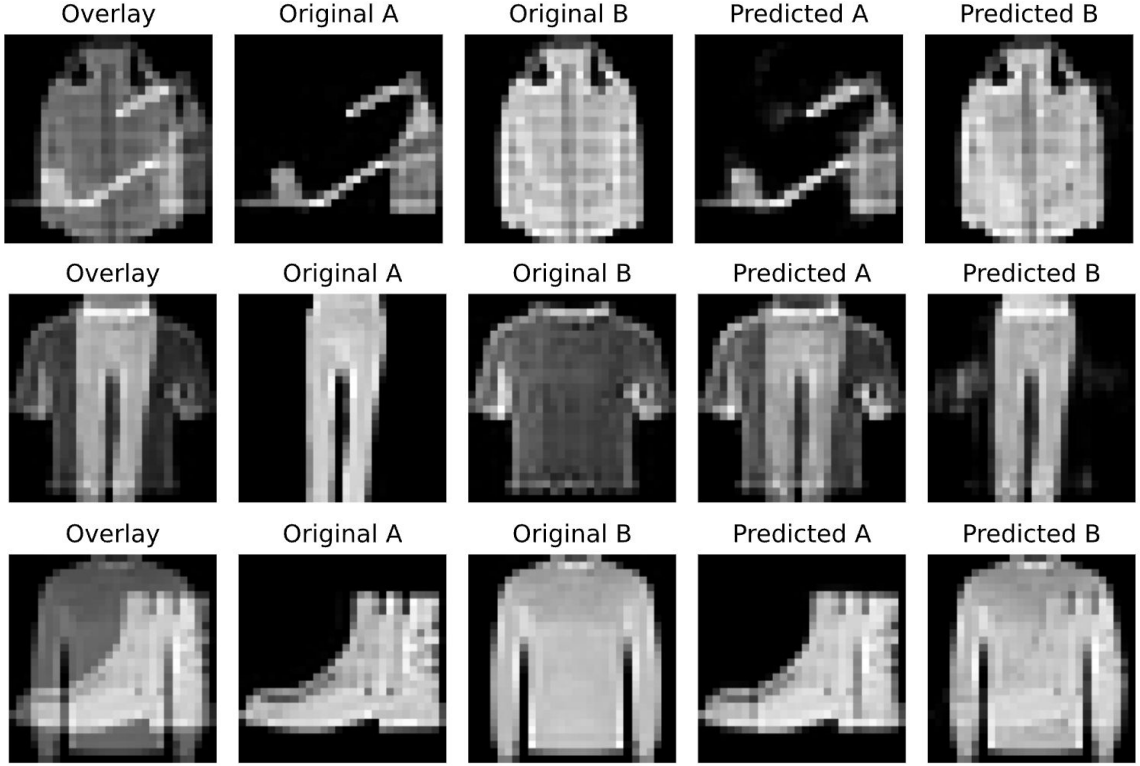
Validation loss stays slightly above training loss

Generalization error does not grow unboundedly

Both losses stay within a relatively small range of overall values



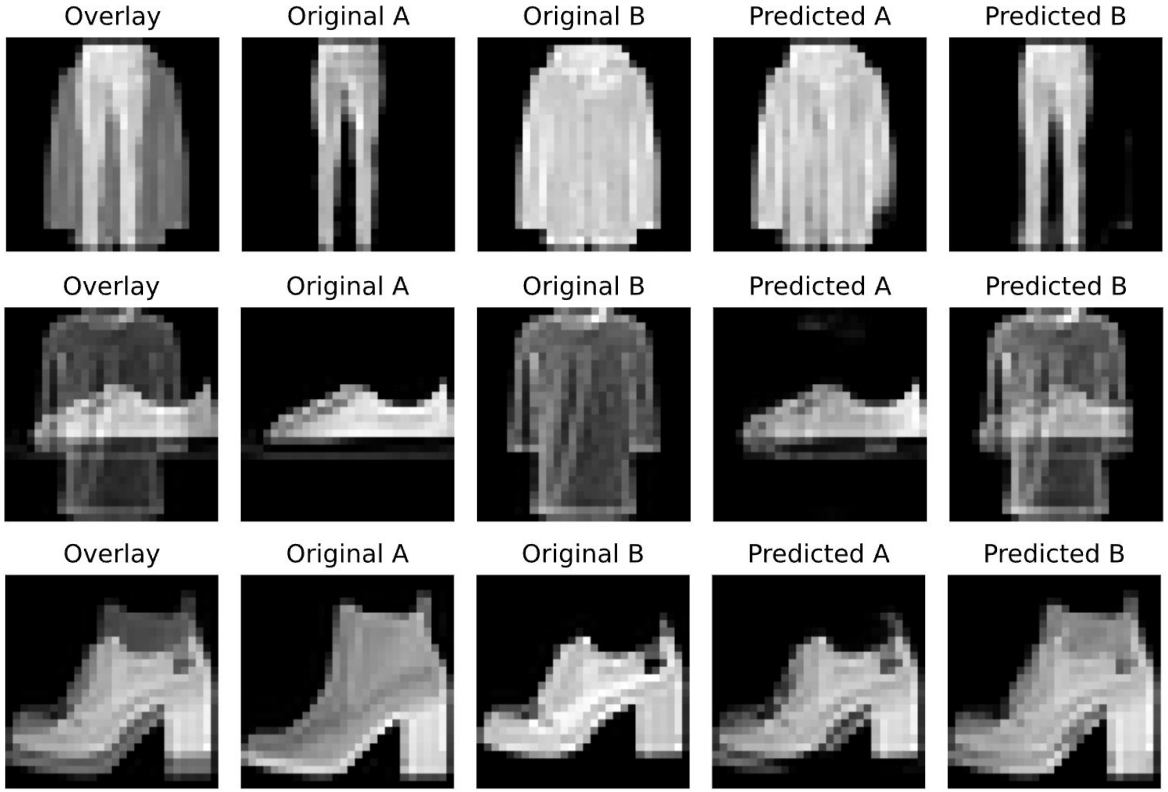
Training Results



Testing

Testing Results

Test Loss:
0.0080



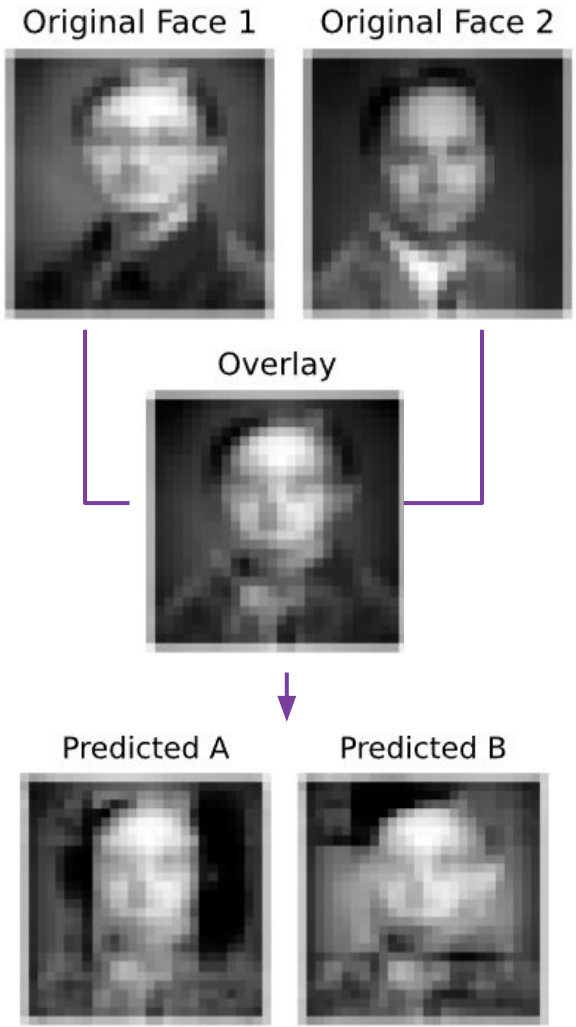
Edge Case: Face Separation?

It does not work :(We tried

Loss: ~0.2 (Test loss within Fashion MNIST was ~0.008)

Man-made horrors beyond comprehension

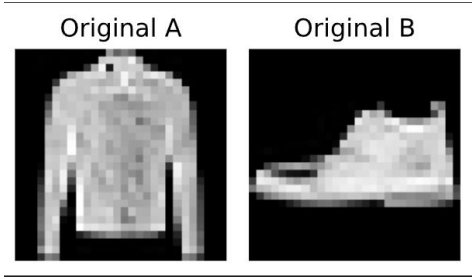
Some of the artifacts may hint at the building blocks the NN actually uses to construct images

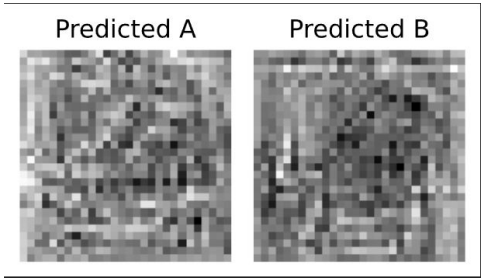


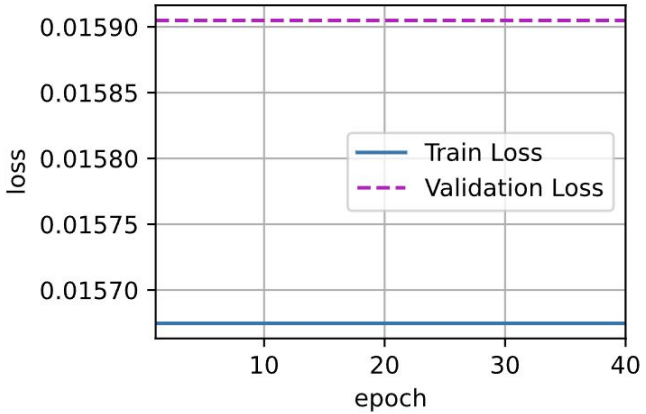
Challenges

Loss Functions

- Tried L2 loss initially, and while the numerical values were better, image prediction was worse
- Cosine similarity was attempted, but did not work, possibly related to only measuring angles and not magnitudes between images *(see figures to right)*
- Other function from image processing, structural similarity index measure (SSIM), was tested, but it did not outperform L1

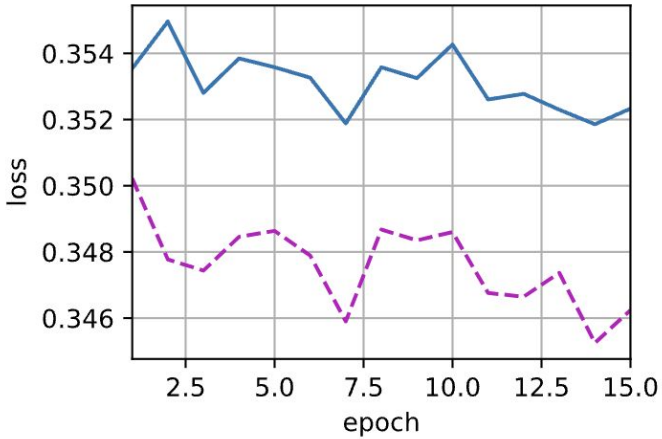


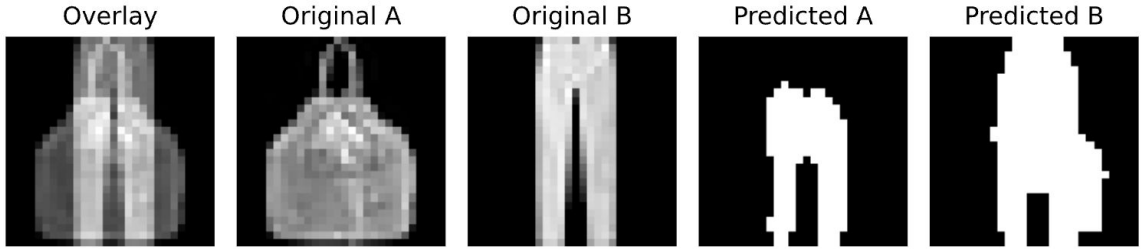




Validation Loop

- Our initial attempt to add the validation loop was... unsuccessful
  - Not really sure what happened here, as the training loop on its own was fine, ended up having to abandon the original notebook





Takeaways

What We Learned

- This model works quite well for Fashion MNIST, but ONLY for Fashion MNIST
- Even when images of professors were compressed to be the same input values as Fashion MNIST, the model still failed spectacularly
- The way that our model finds features seems to be focused on edge separation and requires a high contrast between the background and objects
- Future steps would focus on improving separation of small scale features and textures on images

Questions?