

CS 449

Homework 2

Group 14: Akash Vikram Shroff, Alexis Diaz-Waterman, Seung Hee Daniel Lee, Matthew Britt-Webb

1. (5.0 points) Implement and train your autoencoder on the subset of the Emoji dataset that you selected and augmented:

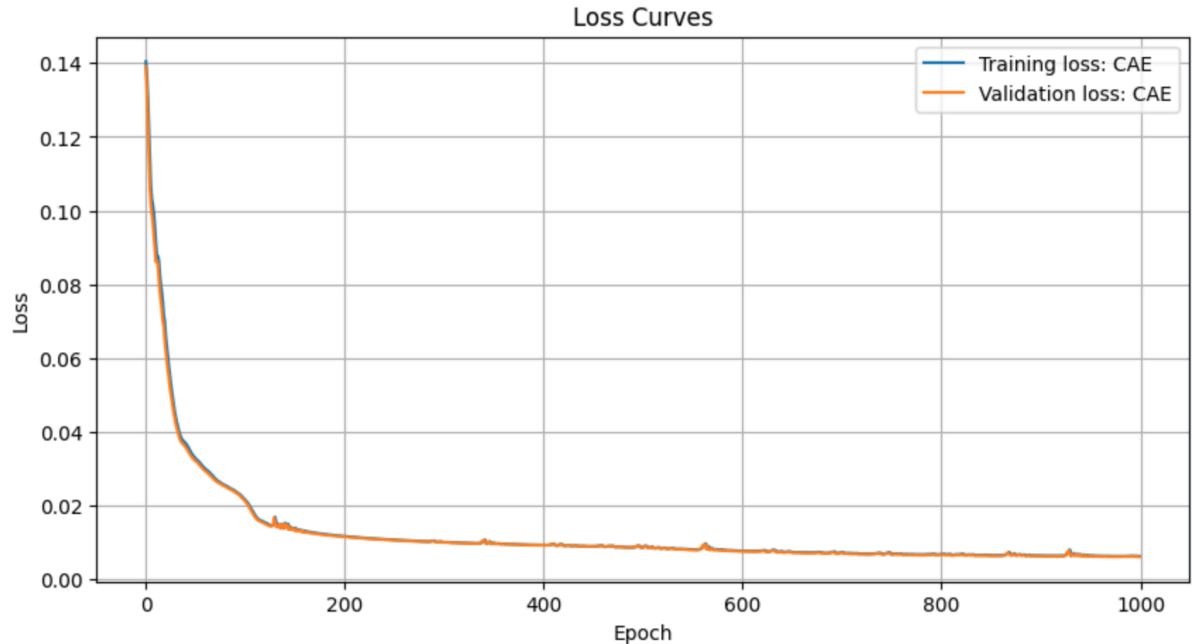
- a. describe your dataset and the steps you used to create it
 - Our data is the subset of the emoji dataset containing all emojis with the substring “man”, such as “woman superhero”, “man supervillain”, etc. The initial dataset contained 610 examples, which we augmented to 1220 examples by randomly rotating, mirroring or flipping each emoji in our subset. Emojis were also resized to 64x64 bytes to manage compute resources.
- b. provide a summary of your architecture (see Adversarial Examples Notebook)

```
ConvAutoEncoder(  
  (encoder): Sequential(  
    (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (1): ReLU()  
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
    (4): ReLU()  
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
  )  
  (decoder): Sequential(  
    (0): ConvTranspose2d(32, 16, kernel_size=(2, 2), stride=(2, 2))  
    (1): ReLU()  
    (2): ConvTranspose2d(16, 3, kernel_size=(2, 2), stride=(2, 2))  
    (3): Sigmoid()  
  )  
)
```

- c. discuss and explain your design choices
 - For our autoencoder, we chose to use a convolutional structure to accurately extract visual features from the emoji images and create embeddings from the images.
 - The encoder has two convolutional layers that are used to extract features from the emoji images - the first one generates 16 feature maps and then the second doubles this to 32. Max pooling layers are used to reduce spatial dimensions and reduce computational load. The ReLU activation function is used to model non-linear relationships between the data.
- d. list hyper-parameters used in the model
 - The hyper-parameters used (apart from model architecture) were num epochs (1000), the learning rate (0.0075) and a scheduler (steplr with step size 100 and gamma 0.1) as well as the gradient descent mechanism (batch, mini-batch, stochastic). All chosen hyperparameters were finetuned against the validation

set and we chose to employ batch gradient descent owing to the relatively small size of the training set.

e. plot learning curves for training and validation loss as a function of training epochs



- P.S. The training and validation loss follow incredibly similar trend patterns and are almost identical. This could perhaps mean that the augmentations on the original dataset did not change enough and some experimentation with adding noise could be fruitful.

f. provide the final average error of your autoencoder on your test set

- The final average error (MSE Loss) of our autoencoder on our test set was **0.00488**

g. discuss any decisions or observations that you find relevant

- By adding more epochs, we noticed that the training and validation loss kept consistently decreasing although it became more marginal as we passed 200 epochs. We originally had images that were not consistent in terms of color but this was resolved as higher epochs were used. Overall, we are still struggling with pixelation issues but this is a tradeoff that has to be made to avoid overfitting - we could have potentially resolved this issue by adding more convolutional layers, however, that resulted in a high degree of overfitting owing to a relatively small dataset.

2. (5.0 points) Separate your dataset into two or more classes using Emoji descriptions and assign labels. Repeat Step 1 adding image classification as an auxiliary task to MSE with a of your choosing. You can choose any classification technique.

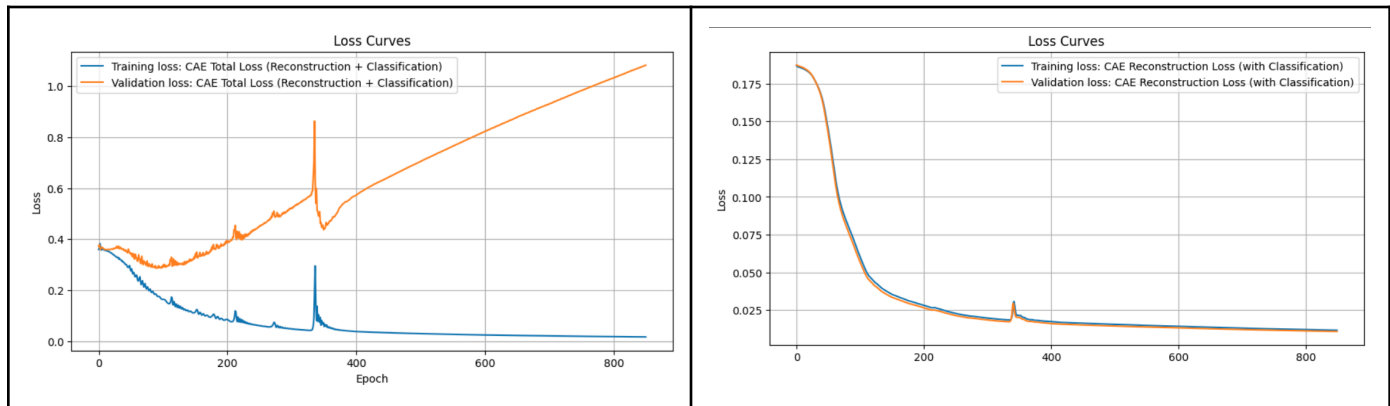
a. describe how you separated your dataset into classes

- We separated our dataset by creating classes for “man” and “woman”. This was done by iterating over the words in each emoji description and counting the number of times “man”, “woman”, “girl” and “boy” appeared. Any emoji name containing exactly one instance of “man” was classified as a man and any emoji name containing exactly one instance of “woman” was classified as a woman. Emojis with multiple people were thus excluded. We then expanded this dataset using the data augmentation (randomly rotating, mirroring or flipping) highlighted in Part 1.

b. describe your classification technique and hyper-parameters

- The classification was done by adding a fully connected layer to the encoder of the CAE (effectively creating a convolutional neural network). We then used the binary cross entropy loss function to measure classification loss and gradient descent was done against the total loss (reconstruction + λ * classification loss).
- The hyper-parameters here were again lr (0.001), num epochs (850) and perhaps most importantly, λ (0.25), or the importance given to the auxiliary task. All of these were finetuned against the validation set.

c. plot learning curves for training and validation loss for MSE and classification Accuracy



d. discuss how incorporating classification as an auxiliary task impacts the performance of your autoencoder

- As you can see from the graphs above, the loss curve for only reconstruction consistently decreases however the total loss grows linearly for the validation set. This can be attributed to tremendous overfitting on the classification task - which is perhaps explained by the relatively simple dataset and similarity between the two classes (several man and woman emojis differ by a very small ponytail). The loss curves for reconstruction are similar to that of the traditional CAE.



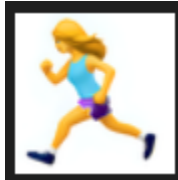
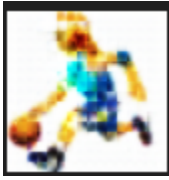
- The average error on the test set is **0.0123**. This is worse than the performance of the traditional CAE - this could be attributed to the fact that the classification task distracted from the image reconstruction (perhaps owing to too high of a lambda, but I noticed a similar result on other lambdas).

e. speculate why performance changed and recommend (but do not implement) an experiment to confirm or reject your speculation

- The reduction in performance could be attributed to the auxiliary task *distracting* from the image reconstruction task and reducing performance. We can verify this by perhaps manually checking the gradients during the training - too high changes coming from the auxiliary task might indicate that one task is taking over the training performance.
- Another potential issue is with our chosen dataset - perhaps by choosing some more distinct emoji classes, we could make the auxiliary task easier and avoid potentially overfitting the auxiliary task which might be making the image reconstruction worse. It might make sense to use similar architecture on two different dataset classes and then compare the results.

3. (5.0 points) Select an attribute from the Emoji dataset (internal or external to your selected subset) to compose with any image from your selected subset. Use vector arithmetic on latent representations to generate a composite image that expresses the attribute.

a. specify which attribute you selected, the vector arithmetic applied and the resulting image(s) as displayed above,

	-		+		=	
Man with ball		Man Running		Woman Running		Woman running with ball

- Man with ball - Man Running + Woman Running = Woman Running with Ball
- We gathered the image arrays of each of our chosen emojis, encoded each image with the model we had already trained, added the encode tensors, and then decoded the composite tensor, again using our trained model

b. provide a qualitative evaluation of your composite image, and

- Our model seems to have kept features of the original “man with ball” image that seemed distinct such as the ball, the elbow pad, and the headband. It also seems to have included the colour of the shirt on “woman running”

c. discuss ways to improve the quality of your generated image.

- The generated image is pixelated and lacks clarity in its features. This might be because the autoencoder is not learning fine enough features or the latent representation is too simplistic and doesn't have enough dimensions. The former might not be the case since reconstructing images seems to maintain most features. We could perhaps increase the latent dimensions. Moreover, we could experiment with the augmentation features - add gaussian noise, and alter image brightness. This would force the autoencoder to wade through more noise and learn more abstract details about the images themselves.