Part 1: MSE Autoencoder

a. Describe your dataset and the steps that you used to create it

The dataset is derived from the Hugging Face "valhalla/emoji-dataset," which contains images of emojis with corresponding text descriptions.
The preprocessing steps include:

1. Filtering: Extract images containing the keyword "face" in the text descriptions. [apx: 200 imgs]
2. Dataset Wrapping: A EmojiDataset class was implemented to manage data loading and transformations.
3. Data Augmentation: Applied transformations such as random horizontal flip, rotation, and color jitter to increase dataset diversity. Each image was augmented 4 times. [apx: 800 imgs]
4. Splitting: The final dataset was split into train (60%), validation (20%), and test (20%) sets.
5. DataLoaders: The processed datasets were wrapped into PyTorch DataLoader instances with batch size 64.

Initially, I was trying to use torchattaks to generate adversarial examples, but I was unable to get it to work. The augmented dataset had some missing values. The text and label columns (only 204 non-null values), while the image column contains 408 non-null entries. Thus I switched to transform based augmentation. In future, I would like to explore the use of torchattaks for generating adversarial examples. And understanding the reason for the missing values in the dataset.

b. Provide a summary of your architecture

The model is a Convolutional Autoencoder with an encoder-decoder structure:

1. Encoder:
    1. 4 convolutional layers with increasing channels (3 → 32 → 64 → 128 → 256).
    2. Kernel size = 3x3, Stride = 2, Padding = 1.
    3. ReLU activation after each convolution.
2. Latent Space:
    1. Fully connected layer compresses the feature map to a 128-dimensional latent vector.
    2. Dropout (p=0.2) is applied for regularization.

3. Decoder:
    1. Mirrors the encoder with ConvTranspose2d layers to reconstruct the original 64x64 image.
    2. Final activation: Sigmoid to constrain output between [0,1].

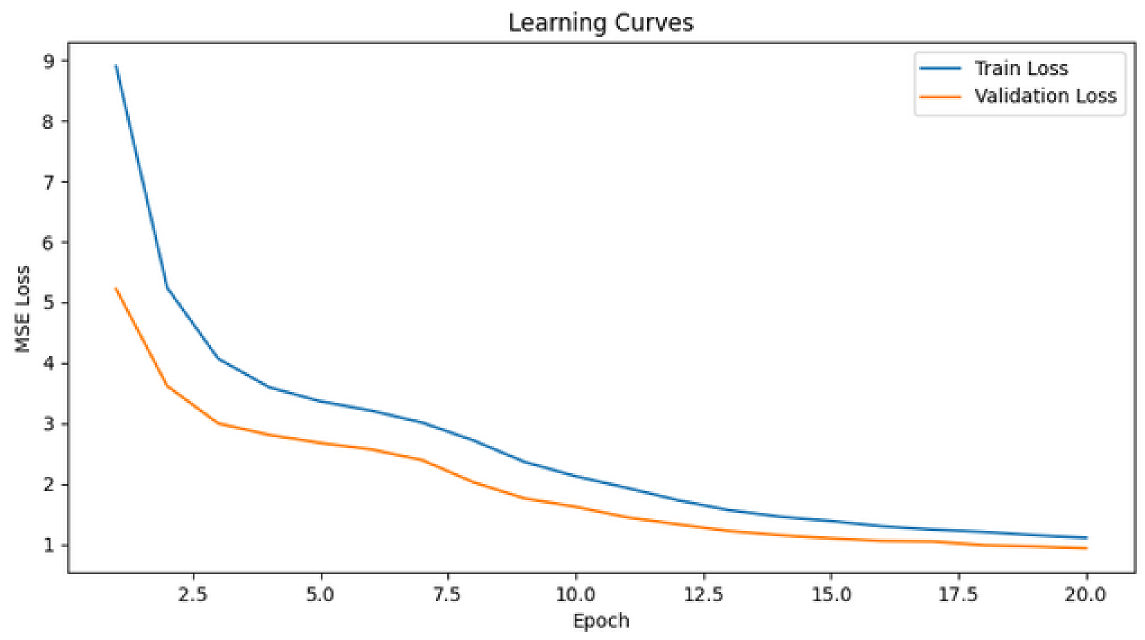
c. Discuss and explain your design choices

- Conv Layers with Strided Convolutions: Reduce spatial dimensions without max-pooling, preventing information loss.
- Latent Dimension = 128: Balances compression and expressiveness.
- Dropout (0.2) in Latent Layer: Prevents overfitting in feature representation (regularisation).
- ReLU Activation: Used in the encoder and decoder (except last layer) for non-linearity.
- Sigmoid in Final Layer: Ensures pixel values remain in a valid image range.
- Adam Optimizer (LR = 1e-3, Weight Decay = 1e-5): Provides stable convergence with L2 regularization.

d. list hyper-parameters used in the model,

1. number of layers in encoder and decoder
2. size of each layer.
3. stride, kernel size, padding of each of them.
4. regularization: dropout.
5. probability of dropout = 0.2.
6. latenet dimension.
7. activation function.: ReLU.
8. learning rate.: 1e-3
9. weight decay: 1e-5
10. num_epochs = 30
11. batch size = 64
12. loss function: MSE

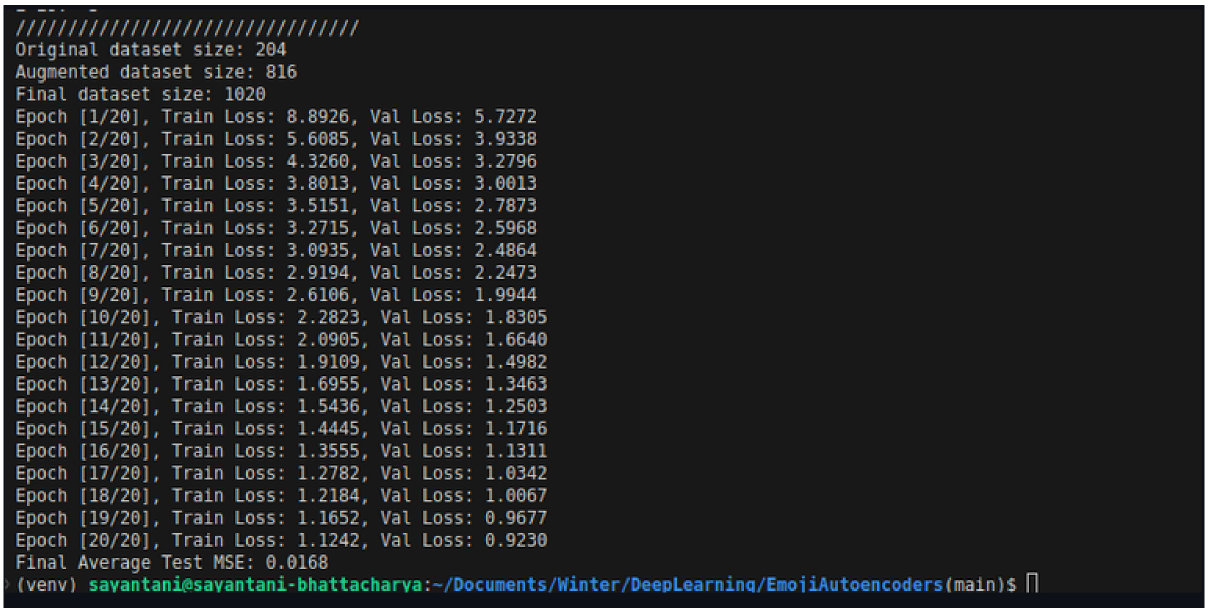
e. plot learning curves for training and validation loss as a function of training epochs,



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

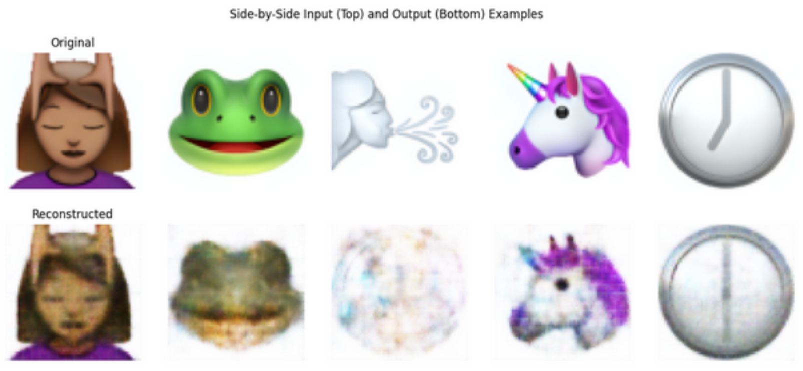
```
Final average MSE error(testset): 0.0168
```

Alt text



```
////////////////////////////////
Original dataset size: 204
Augmented dataset size: 816
Final dataset size: 1020
Epoch [1/20], Train Loss: 8.8926, Val Loss: 5.7272
Epoch [2/20], Train Loss: 5.6085, Val Loss: 3.9338
Epoch [3/20], Train Loss: 4.3260, Val Loss: 3.2796
Epoch [4/20], Train Loss: 3.8013, Val Loss: 3.0013
Epoch [5/20], Train Loss: 3.5151, Val Loss: 2.7873
Epoch [6/20], Train Loss: 3.2715, Val Loss: 2.5968
Epoch [7/20], Train Loss: 3.0935, Val Loss: 2.4864
Epoch [8/20], Train Loss: 2.9194, Val Loss: 2.2473
Epoch [9/20], Train Loss: 2.6106, Val Loss: 1.9944
Epoch [10/20], Train Loss: 2.2823, Val Loss: 1.8305
Epoch [11/20], Train Loss: 2.0905, Val Loss: 1.6640
Epoch [12/20], Train Loss: 1.9109, Val Loss: 1.4982
Epoch [13/20], Train Loss: 1.6955, Val Loss: 1.3463
Epoch [14/20], Train Loss: 1.5436, Val Loss: 1.2503
Epoch [15/20], Train Loss: 1.4445, Val Loss: 1.1716
Epoch [16/20], Train Loss: 1.3555, Val Loss: 1.1311
Epoch [17/20], Train Loss: 1.2782, Val Loss: 1.0342
Epoch [18/20], Train Loss: 1.2184, Val Loss: 1.0067
Epoch [19/20], Train Loss: 1.1652, Val Loss: 0.9677
Epoch [20/20], Train Loss: 1.1242, Val Loss: 0.9230
Final Average Test MSE: 0.0168
(venv) sayantani@sayantani-bhattacharya:~/Documents/Winter/DeepLearning/EmojiAutoencoders(main)$
```

g. provide a side-by-side example of 5 input and output images

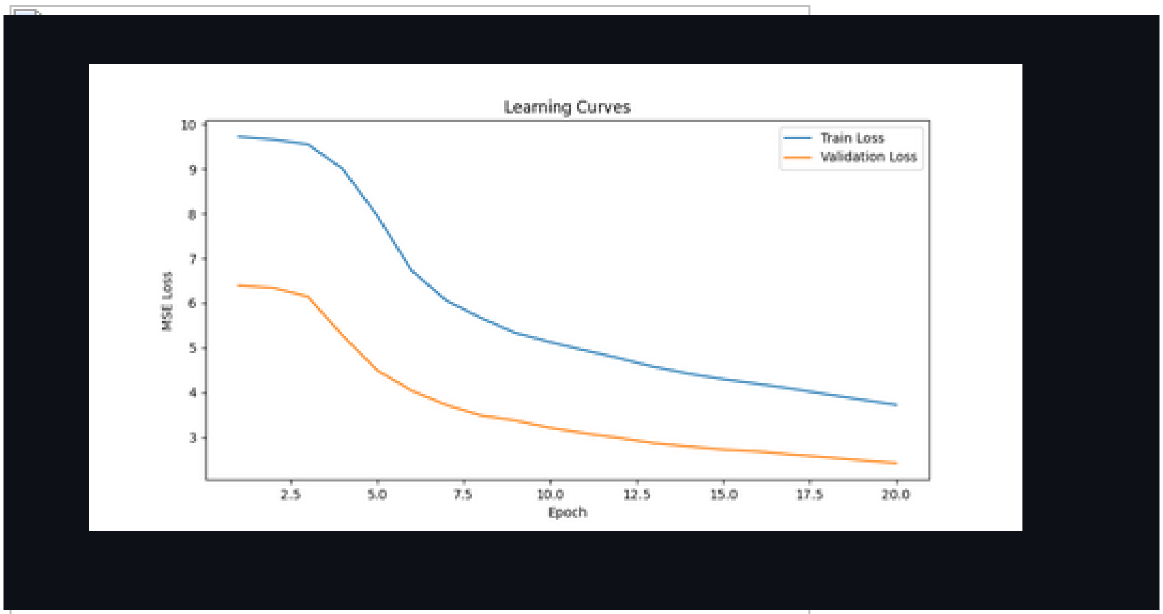


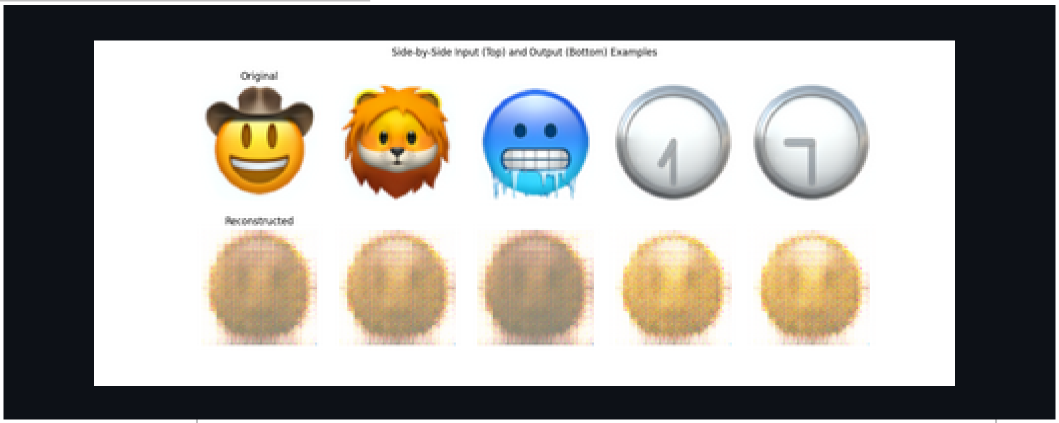
Alt text

h. discuss any decisions or observations that you find relevant.

1. Augmentation Boosted Performance: Increased dataset size significantly, improving generalization.

Plot and image without adding data augmentation:





2. Training vs Validation Loss: Consistently decreasing, indicating the model is learning meaningful features.
3. Final Test MSE: Achieved a stable reconstruction quality, indicating effective encoding.

Part 2: Encoder-Classifier Model

a. describe how you separated your dataset into classes

The dataset is derived from the Hugging Face Emoji dataset, and filtering is done based on whether the word "face" appears in the text description. Then, a class mapping is applied to categorize images into two classes:

1. Class 0 (Happy Faces): Includes descriptions like "happy face," "grinning face," and "smiling face."
2. Class 1 (Sad Faces): Includes descriptions like "sad face," "crying face," and "frowning face."

Images with descriptions that do not match any predefined class are discarded. The dataset is then augmented by generating multiple transformed copies of each image using horizontal flips, rotations, and color jittering. The final dataset is split into 60% training, 20% validation, and 20% test sets.

b. Describe your classification technique and hyper-parameters

The model is a multi-task autoencoder that simultaneously reconstructs input images and classifies them into happy or
sad categories. The encoder compresses the image into a latent representation, which is then used for both reconstruction and classification.

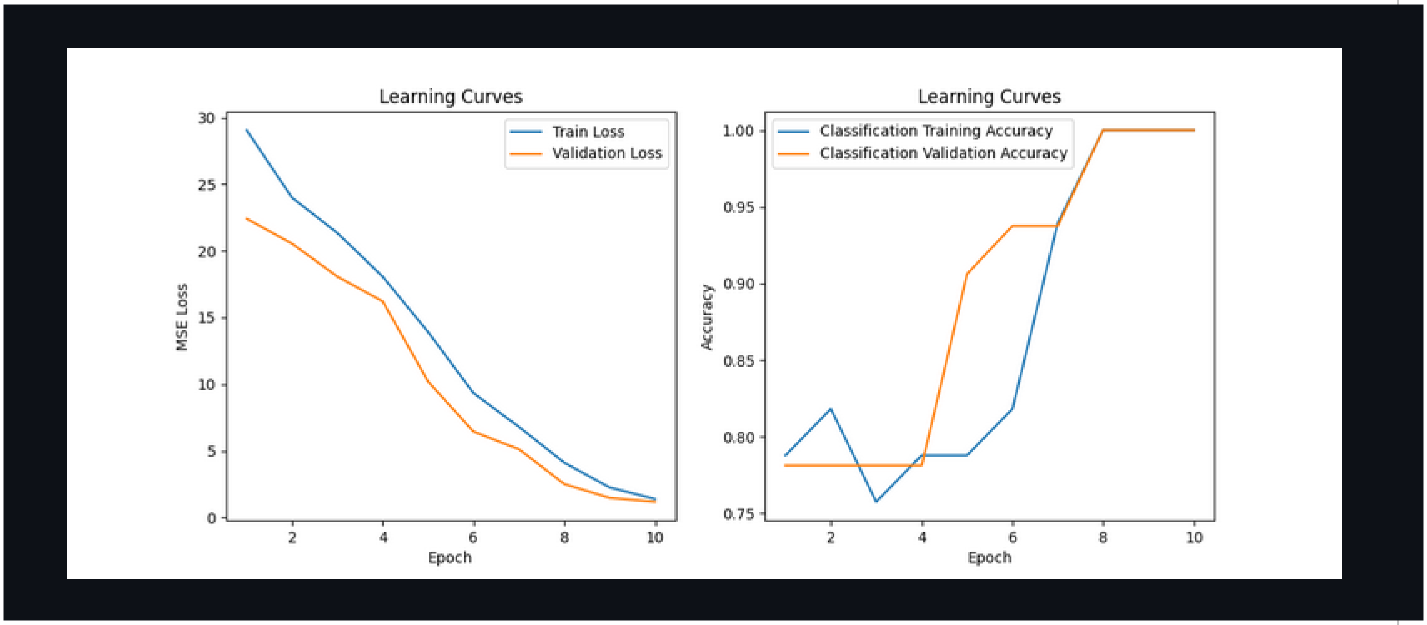
Key hyperparameters:

1. Autoencoder Architecture:
    1. Convolutional encoder with 4 layers, ReLU activation, and dropout (p=0.2)
    2. Fully connected layers map to a latent dimension of 128
    3. Decoder mirrors the encoder with transposed convolutions
2. Classification:
    1. Fully connected layer mapping the latent space to 2 class logits
    2. Uses CrossEntropyLoss
3. Training Setup:
    1. Optimizer: Adam with learning rate 0.001
    2. Batch size: 64
    3. Training epochs: 10 (originally planned for 20)
    4. Loss Function: Weighted combination of autoencoder loss (MSELoss) and classification loss (CrossEntropyLoss)
    5. Weighting factor for classification loss: lambda_classification = 0.5

c. Plot learning curves for training and validation loss for MSE and

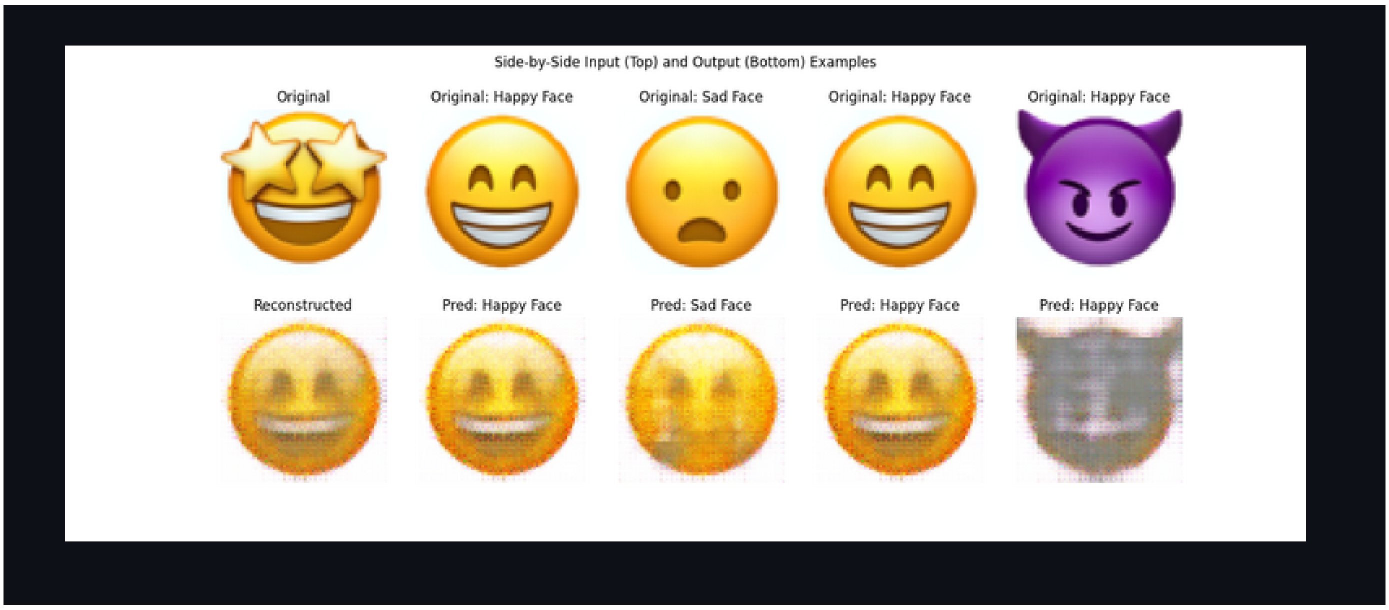
classification accuracy

Alt text



d. provide a side-by-side example of 5 input and output images

Alt text

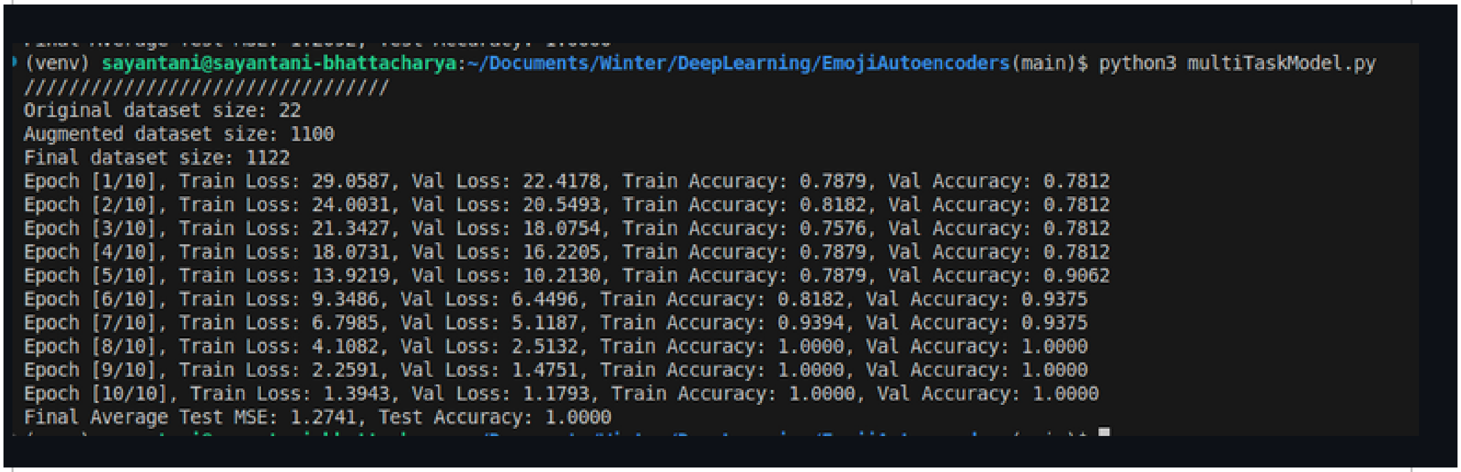


e. discuss how incorporating classification as an auxiliary tasks impacts the performance of your autoencoder

Because of competing objectives, the autoencoder attempts to preserve as much information as possible for reconstruction, while classification might push it to focus on class-specific features, thus potentially reducing reconstruction quality. The model is hyperparameter Sensitive. The weight lambda_classification needs careful tuning—too high, and reconstruction quality suffers; too low, and classification has little impact. In the current case, there was basrely any data avaivlable for training, so I had to add 50 each
augmented image to each element of the dataset. This resulted in overfitting, as you can see from the training and validation loss curves and tranining epochs. But, I belive this kind a model, also forced the

encoder to learn features that are not just useful for reconstruction but also discriminative for classification, leading to a more structured latent space. And given the right amount of data, this could lead to better generalization.

Alt text



```
(venv) sayantani@sayantani-bhattacharya:~/Documents/Winter/DeepLearning/EmojiAutoencoders(main)$ python3 multiTaskModel.py
/////////////////////////////////
Original dataset size: 22
Augmented dataset size: 1100
Final dataset size: 1122
Epoch [1/10], Train Loss: 29.0587, Val Loss: 22.4178, Train Accuracy: 0.7879, Val Accuracy: 0.7812
Epoch [2/10], Train Loss: 24.0031, Val Loss: 20.5493, Train Accuracy: 0.8182, Val Accuracy: 0.7812
Epoch [3/10], Train Loss: 21.3427, Val Loss: 18.0754, Train Accuracy: 0.7576, Val Accuracy: 0.7812
Epoch [4/10], Train Loss: 18.0731, Val Loss: 16.2205, Train Accuracy: 0.7879, Val Accuracy: 0.7812
Epoch [5/10], Train Loss: 13.9219, Val Loss: 10.2130, Train Accuracy: 0.7879, Val Accuracy: 0.9062
Epoch [6/10], Train Loss: 9.3486, Val Loss: 6.4496, Train Accuracy: 0.8182, Val Accuracy: 0.9375
Epoch [7/10], Train Loss: 6.7985, Val Loss: 5.1187, Train Accuracy: 0.9394, Val Accuracy: 0.9375
Epoch [8/10], Train Loss: 4.1082, Val Loss: 2.5132, Train Accuracy: 1.0000, Val Accuracy: 1.0000
Epoch [9/10], Train Loss: 2.2591, Val Loss: 1.4751, Train Accuracy: 1.0000, Val Accuracy: 1.0000
Epoch [10/10], Train Loss: 1.3943, Val Loss: 1.1793, Train Accuracy: 1.0000, Val Accuracy: 1.0000
Final Average Test MSE: 1.2741, Test Accuracy: 1.0000
```

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

1. The Classification Task is Too Simple:

Speculation: The classification task has only two classes (happy vs. sad), which may not be challenging enough to meaningfully improve the learned representation. A more complex classification task (e.g., multi-class sentiment detection) might lead to more informative latent features. Experiment: Modify the classification task to include more fine-grained emotion categories (e.g., neutral, angry, surprised). If classification accuracy drops initially but improves feature learning in the long run, it indicates that a more complex auxiliary task benefits the autoencoder.

2. Latent Space Bottleneck is Too Small:

Speculation: The latent dimension (128128) may not be sufficient to encode both visual reconstruction and classification features effectively. A smaller latent space forces compression, which may be beneficial for generalization but can limit performance.Experiment: Train models with different latent dimensions (e.g., 64, 128, 256, 512) and observe their impact on both classification accuracy and reconstruction quality. If larger latent spaces improve classification but degrade reconstruction, then the bottleneck hypothesis is valid.

3. Data Augmentation induced noise:
   Speculation: The extensive data augmentation (horizontal flip, rotation, color jitter) may have introduced variations that make reconstruction harder. While these augmentations improve classification robustness, they might add complexity that hurts the autoencoder's ability to learn meaningful features. Experiment: Try using the same model, but with different dataset, which has enough original images, to not need augmentation.

Part 3: Composite Image generator

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

Attribute Selected: The attribute selected for transformation is "heart-shaped eyes," which distinguishes the "smiling face with heart-shaped eyes" emoji from the "grinning face" baseline.

Vector Arithmetic Applied:

1. z_attribute = z_featuredImg - z_baseline
2. z_compositeImg = z_input + z_attribute

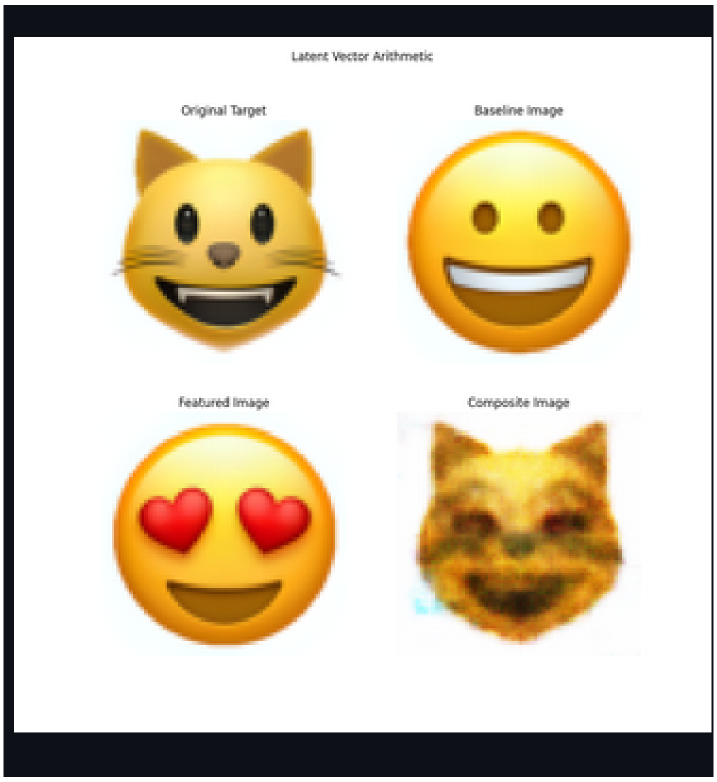
This equation extracts the heart-shaped eyes attribute from the featured image and adds it to the target input image's latent representation.

Resulting Image(s):

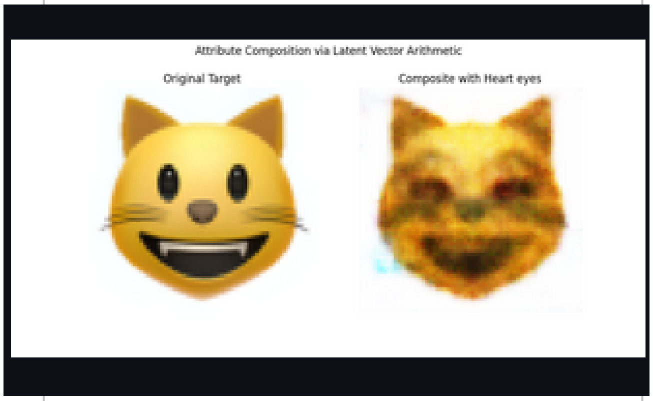
- The original target image: "smiling cat face with open mouth."
- The baseline image: "grinning face."
- The featured image: "smiling face with heart-shaped eyes."
- The composite image, which is expected to resemble the input emoji but now with added heart-shaped eyes.

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

The composite image should ideally retain the cat-like features of the input emoji while incorporating heart-shaped eyes.



Alt text


Attribute Composition via Latent Vector Arithmetic
Original Target
Composite with Heart eyes

As we can observe, the composite image has:
Blurring or distortion in the eye region, with red pixels showing up which shows presence of heart-shaped eye attribute in the image.
There is an incomplete attribute transfer (e.g., the cat's eyes changing but not fully transforming into heart shapes).
Also there are color inconsistencies due to how the latent vectors interact.

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

Several improvements could be applied to enhance the quality of the generated composite image. Some of them are:

1. Better Attribute Isolation:

   Instead of directly subtracting the entire latent vector of the baseline from the featured image, consider using Principal Component Analysis (PCA) to isolate the heart-shaped eye feature more precisely. Train a model to explicitly learn interpretable latent dimensions corresponding to specific attributes.

2. Fine-Tuning with More Data:

   The dataset is too small, so the model struggles with attribute transfer.Using a larger emoji dataset with diverse examples of similar transformations could improve results. Also, better augmentation techniques could be applied to increase the dataset size.

3. Using Interpolation Instead of Direct Arithmetic:

Instead of direct vector subtraction and addition, try a weighted interpolation, that can be tuned to control the strength of the attribute.

4. Post-Processing Enhancements:

   Use GAN-based refinement models to improve attribute blending and enhace final image quality.