CORE

a study into classifying aesthetics
using machine learning

Ashley Zhang

Table of Contents

Ideation

1. Aesthetics
    a. image-to-text: given an image of an article of clothing, generate text describing it
    b. text-to-image: given certain parameters, generate an image of clothing matching said params
    c. classification: given an image, classify image within its aesthetic realm
        i. VGG19 → pre-trained model
        ii. k-nearest neighbors
        iii. decision tree
        iv. support vector machine
2. x-core
    a. -core suffix typically denotes "the central, innermost, or most essential part of anything"
    b. but may also be related to "the permanent, dedicated, and completely faithful nucleus of a group or movement"
    c. x-core being representative of an arbitrary aesthetic
        i. from [Cottagecore, Dreamcore, Normcore, And Other -Core Words](#).

Dataset Creation

- currently experiencing lack of datasets catering towards fashion, particularly that of Gen Z trends
- create own image datasets (things to keep in mind)
    a. train, test, validation 80/10/10 split
    b. labels
        i. trending aesthetics based on Aesthetics Wiki: y2k, coquette, academia, grunge, cottagecore, punk, vintage, dreamcore
    c. size of files
        i. use Google Colab GPU for faster runtime
    d. standardization
        i. if obtaining from internet, need to make <u>same size and extension</u>
        ii. convert to grayscale, get value of each pixel, and store in csv file + label
        iii. use <u>standardscaler</u> to standardize pixel values
    e. website inspo: Pinterest, Aesthetics Wiki, Tumblr
    f. some aesthetics overlap or have similar visuals
        i. label as widest overarching aesthetic / don't go into specifics
        ii. can always fine tune later

Custom Dataset

- tried programmatically downloading using Javascript in inspect element, doesn't work anymore due to updates to Google's HTML/CSS architecture
- using Fatkun Batch Download Image in Chrome Extensions, can download from multiple tabs
    - must manually determine relevancy of images
- crop and resize images to fit VGG19 input dimensions of 224×224
- zip -r x-core.zip . -x ".DS_Store" -x "__MACOSX"
- find . -name '.DS_Store' -type f -delete
    - removes __MACOSX folder and .DS_Store files from zipped file
    - otherwise looping through causes errors
- get images from various sources due to variability in definition of aesthetic

Small-Scale Prototype

1. Data Collection
   a. using Fatkun Batch Download, obtain around 100 images for each of three categories: y2k, grunge, and coquette
      i. manually filter out irrelevant images
      ii. for first prototype, all images were downloaded from first page of Google Images
   b. zip files for upload into Google Colab → consider Jupyter Notebook
      i. use commands above to delete Mac hidden files and folders
2. Preprocessing Data
   a. generate tf.data.Dataset object using image_dataset_from_directory
      i. use image_size parameter to resize images to (224, 224) after reading from disk

```
# data pipeline
data = image_dataset_from_directory('data', image_size=(224, 224))
```

      ii.

```
Found 300 files belonging to 3 classes.
```

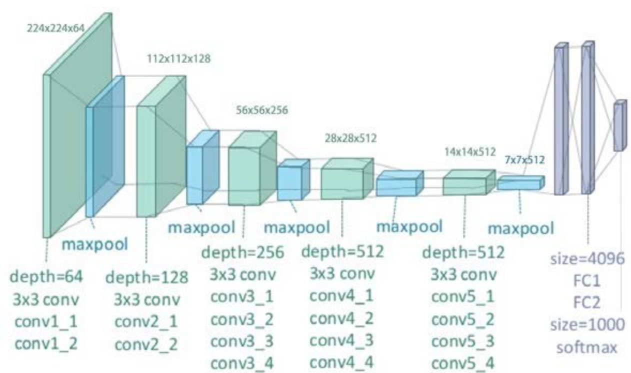
   b. normalize data by dividing x values (rgb of pixels) by 255

```
data = data.map(lambda x, y: (x/255, y))
```

3. Splitting Data
   a. split into train, test, validation using 80/10/10
   b. use take() and skip() methods

```
train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size + val_size).take(test_size)
```

4. Transfer Learning
   a. using pre-trained model VGG19
   b.



      from [Illustration of the network architecture of VGG-19 model]
   c. remove final dense layers → specify 3 classes instead of expected 1,000 as output
   d. define the model using VGG19 as the input and a dense layer of 3 units using softmax activation as the output
      i. softmax: function rescaling numerical input tensors into probabilities (elements within [0, 1] and add to 1)
      ii. normalizes output of model to fit output classes
      iii. ex. [0.0021657, 0.00588697, 0.11824302, 0.87370431]
5. Compiling Model
   a. loss function: sparse categorical cross entropy
      i. since input classes of 3 > 2
      ii. consider one-hot encoding labels → must change to categorical cross entropy

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv4 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv4 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 3)	75267

```
Total params: 20,099,651
Trainable params: 75,267
Non-trainable params: 20,024,384

None
```

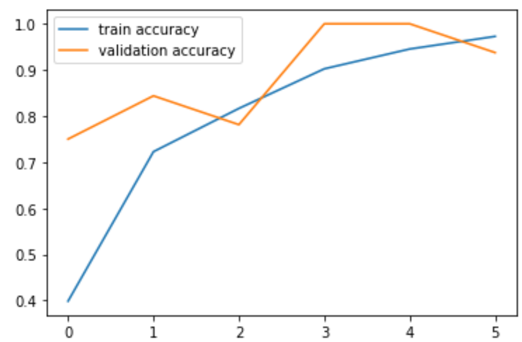
b. optimizer: adam vs. SGD

6. Training
    a. implement early stopping in callbacks to avoid overfitting
    b. fit with 20 epochs and a batch size of 32

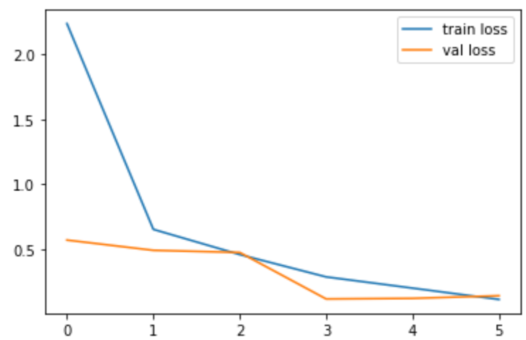
7. Data & Results from Small-Scale Test

```
Epoch 1/20
8/8 [==============================] - 8s 580ms/step - loss: 2.2388 - accuracy: 0.3984 - val_loss: 0.5694 - val_accuracy: 0.7500
Epoch 2/20
8/8 [==============================] - 7s 693ms/step - loss: 0.6521 - accuracy: 0.7227 - val_loss: 0.4905 - val_accuracy: 0.8438
Epoch 3/20
8/8 [==============================] - 9s 899ms/step - loss: 0.4577 - accuracy: 0.8164 - val_loss: 0.4739 - val_accuracy: 0.7812
Epoch 4/20
8/8 [==============================] - 7s 557ms/step - loss: 0.2853 - accuracy: 0.9023 - val_loss: 0.1152 - val_accuracy: 1.0000
Epoch 5/20
8/8 [==============================] - 7s 560ms/step - loss: 0.1988 - accuracy: 0.9453 - val_loss: 0.1198 - val_accuracy: 1.0000
Epoch 6/20
8/8 [==============================] - 7s 615ms/step - loss: 0.1115 - accuracy: 0.9727 - val_loss: 0.1398 - val_accuracy: 0.9375
```

a1. Training Epochs



b1. Train and Validation Accuracy



c1. Train and Validation Loss

```
[[1.         0.         0.        ]
 [0.         1.         0.        ]
 [0.         0.33333333 0.66666667]]
[[3 0 0]
 [0 6 0]
 [0 1 2]]
```

d1. Confusion Matrices

	precision	recall	f1-score	support
0	1.00	1.00	1.00	3
1	0.86	1.00	0.92	6
2	1.00	0.67	0.80	3
accuracy			0.92	12
macro avg	0.95	0.89	0.91	12
weighted avg	0.93	0.92	0.91	12

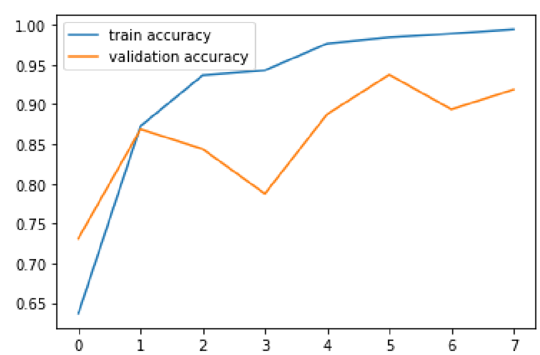
e1. Classification Report

Upscaling

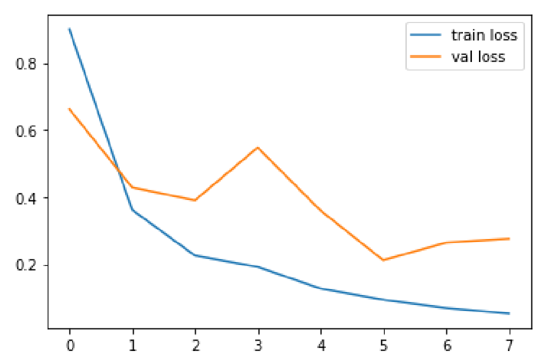
1. Obtaining more data
    a. aiming for around 500 images per class
    b. using Fatkun Batch Download again
    c. sourcing from multiple places this time, particularly from pre-curated Pinterest boards
        i. due to subjectivity of aesthetics, obtaining images from multiple “collections” already created by humans can potentially prevent biases
        ii. particularly in the broad scopes of grunge and y2k, it is evident that people have different definitions of what truly encompasses the aesthetic
    d. End Results
        i. coquette: 524 images +
        ii. grunge: 528 images +
        iii. y2k: 526 images = 1578 images total
2. Same pre-processing, splitting, compiling, and training process and parameters
3. Data & Results from upscaled version

```
Epoch 1/20
40/40 [==============================] - 29s 632ms/step - loss: 0.9002 - accuracy: 0.6367 - val_loss: 0.6622 - val_accuracy: 0.7312
Epoch 2/20
40/40 [==============================] - 27s 645ms/step - loss: 0.3620 - accuracy: 0.8727 - val_loss: 0.4288 - val_accuracy: 0.8687
Epoch 3/20
40/40 [==============================] - 27s 630ms/step - loss: 0.2259 - accuracy: 0.9367 - val_loss: 0.3903 - val_accuracy: 0.8438
Epoch 4/20
40/40 [==============================] - 28s 659ms/step - loss: 0.1920 - accuracy: 0.9430 - val_loss: 0.5478 - val_accuracy: 0.7875
Epoch 5/20
40/40 [==============================] - 28s 625ms/step - loss: 0.1277 - accuracy: 0.9766 - val_loss: 0.3602 - val_accuracy: 0.8875
Epoch 6/20
40/40 [==============================] - 27s 636ms/step - loss: 0.0942 - accuracy: 0.9844 - val_loss: 0.2121 - val_accuracy: 0.9375
Epoch 7/20
40/40 [==============================] - 27s 633ms/step - loss: 0.0695 - accuracy: 0.9891 - val_loss: 0.2644 - val_accuracy: 0.8938
Epoch 8/20
40/40 [==============================] - 27s 617ms/step - loss: 0.0531 - accuracy: 0.9945 - val_loss: 0.2761 - val_accuracy: 0.9187
```

a2. Training Epochs



b2. Train and Validation Accuracy



c2. Train and Validation Loss

```
[[0.91304348 0.         0.08695652]
 [0.0212766  0.93617021 0.04255319]
 [0.02222222 0.04444444 0.93333333]]
[[42  0  4]
 [ 1 44  2]
 [ 1  2 42]]
```

d2. Confusion Matrices

```
              precision    recall  f1-score   support

           0       0.95      0.91      0.93        46
           1       0.96      0.94      0.95        47
           2       0.88      0.93      0.90        45

    accuracy                           0.93       138
   macro avg       0.93      0.93      0.93       138
weighted avg       0.93      0.93      0.93       138
```

e2. Classification Report

f1. Predicted versus Actual outputs where 0 : coquette :: 1 : grunge :: 2 : y2k

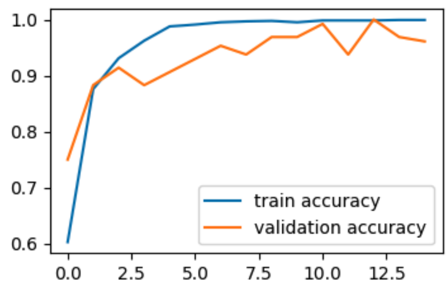
4. Optimization
    a. https://www.kaggle.com/questions-and-answers/279139
    b. reducing validation loss and increasing validation accuracy
        i. decreasing batch size from 32 to 20 yields around 15% val loss and 95% val accuracy → val loss considerably lower but also takes more time to train

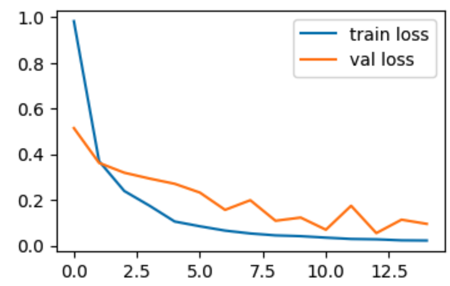
One-hot Encoding Labels

1. using tf.one_hot and map() transform all labels in tf.dataset object
2. will increase accuracy after upscaling
3. changed loss function from sparse categorical cross entropy to categorical cross entropy
4. changed accuracy metric from accuracy to categorical accuracy
5. Data & Results from one-hot encoded version

```
Epoch 15/20
38/38 [==============================] - 47s 1s/step - loss: 0.0224 - categorical_accuracy: 0.9992 - val_loss: 0.0953 - val_categorical_accuracy: 0.9609
```

a3. Training Epochs





b3. Train and Validation Loss

c3. Train and Validation Accuracy

```
[[0.97560976 0.         0.02439024]
 [0.02702703 0.94594595 0.02702703]
 [0.         0.04       0.96      ]]
[[40  0  1]
 [ 1 35  1]
 [ 0  2 48]]
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        41
           1       0.95      0.95      0.95        37
           2       0.96      0.96      0.96        50

    accuracy                           0.96       128
   macro avg       0.96      0.96      0.96       128
weighted avg       0.96      0.96      0.96       128
```

d3. Confusion Matrices

e3. Classification Report

f2. Predicted versus Actual outputs where `0 : coquette :: 1 : grunge :: 2 : y2k`

6. performance overall improved, generating 0.0972 loss and 0.984 accuracy when evaluating model on testing dataset

```
model.evaluate(test, batch_size = 32)
✓ 7.5s
4/4 [==============================] - 7s 709ms/step - loss: 0.0972 - categorical_accuracy: 0.9844

[0.09721124172210693, 0.984375]
```