AI6126 Advanced Computer Vision

Homework Assignment 1

School of Computer Science and Engineering

Programme: MSAI

Date: 29 March 2024

Authored By: Tan Jie Heng Alfred (G2304193L)

Contents

1. Introduction

In this project, we attempted to identify and classify attributes depicted within a fashion photograph. The dataset consists of 6000 images split into 5000 for training and 1000 for validation. There are a total of 6 major categories, which could be further subdivided into 26 attribute labels. However, there is an imbalance in the dataset (see Figure 1) – significantly many images have only some attributes. For instance, most of the training images have attribute `solid` for `class 1`. As such, this is an imbalanced multi-label classification problem, as we attempt to construct a model that predict which attributes a given image has.

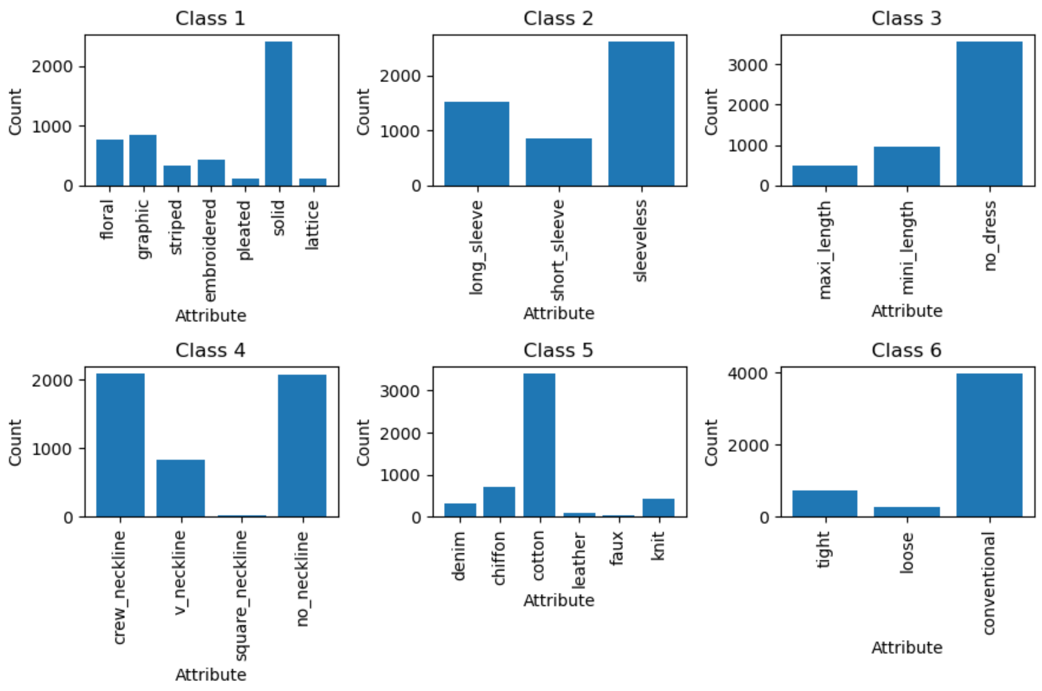

Figure 1: Distribution of label attributes in all the classes of images in training set

2. Model Selection and Implementation

Since we expect a large imbalance in the dataset, we naively predicted the majority class attribute in each class using both the training and validation dataset, as a baseline. Specifically, we obtained the label `[5 2 2 3 2 2]` where the value v_i in the i^{th} index corresponds to v_i^{th} attribute in the i^{th} class. We evaluated this against the test set, obtaining an average per-class accuracy of 0.2308. This metric can be obtained by calculating the overall accuracy for each class label and averaging them.

In our search for a better, learnable model, we explored the use of pretrained models such as EfficientNet-S, Swin Transformer-T, and Visual Attention Network-T (VAN-T). We ultimately decided on VAN-T that was pre-trained on ImageNet-1000, primarily due to its lighter weight nature. With roughly 4.11M parameters, VAN-T has less than a quarter of the parameters that EfficientNet-S and Swin Transformer-T have. This comes at a poorer performance on the ImageNet-1000 dataset, where VAN-T has a 93.02% Top-5 accuracy as compared to 96.878% and 96.132% for EfficientNet-S and Swin-T respectively. Although there is a significant difference in performance, the smaller size of VAN-T makes it more feasible for us to fine-tune the pretrained model on our dataset.

VAN was first proposed by [3], incorporating a linear attention mechanism by using the Linear Kernal Attention (LKA) module,

$$\text{Attention} = \text{Conv}_{1\times 1}\left(\text{DWDC}\big(\text{DWC}(F)\big)\right)$$
$$\text{Output} = \text{Attention} \otimes F$$

where F is the input feature, DW-Conv refers to depthwise convolution, DWDC refers to dilated depthwise convolution, $\text{Conv}_{1\times 1}$ refers to 1×1 pointwise convolution, and the operator $\otimes$ is an element-wise product. This was done to decompose large kernel convolution, while mimicking the self-attention mechanism. The authors argued that LKA takes into account the local structure information (similar to convolutions), long-range dependencies (similar to self-attention), while incorporating channel and spatial adaptabilities.

For the pretrained VAN-T, we obtained it from the open-source `mmpretrain` library from Open-MMLab [4]. Because it was pretrained on a different dataset, we will only use the feature extractor portion (i.e., exclude classifier head) of the VAN, and train a separate classifier head for our classification task. Our classifier head is a dense linear feedforward network consisting of two hidden layers with dropout ($p = 0.3$) for each hidden layer as a regularization technique to reduce overfitting. For the output layer, we have one output head per class, each with varying number of output nodes depending on the number of attributes in that class. Then, we apply a softmax activation function across each head, before concatenating them into a 26-dimensional vector as our final output classification.

For this supervised learning task, we converted the labels into a 26-dimensional one-hot vector, matching the size of our model output. Then, to address the imbalance dataset issue, we implemented an asymmetric loss [2], which allows the network to focus on the hard negatives while maintaining the contribution of positive samples. This is possible by assigning different exponential decay factors for positive and negative samples (γ_+ and γ_-), while thresholding the probabilities of negative samples such that we discard the very 'easy' negative samples (i.e., those with very low probabilities). Putting together, we have the asymmetric loss (ASL) to be,

$$\text{ASL} = \begin{cases} L_+ = (1-p)^{\gamma_+}\log(p) \\ L_- = (p_m)^{\gamma_-}\log(1-p_m) \end{cases},$$

where L_+ and L_- correspond to the loss for positive and negative labels respectively, p is the prediction from the model, and $p_m = \max(p - m, 0)$, is the thresholding of the negative samples, for some threshold m. With a larger γ_- (or γ_+), we give less importance to the negative (or positive) labels, as L_- (or L_+) becomes smaller. Since all our predictions only consists of 6 positive attributes, we would want to give less importance to the more frequent negative predictions. As such, we choose $\gamma_- = 4 > 1 = \gamma_+$. For our threshold, we have $m = 0.05$, ignoring very 'easy' negatives.

The original code implementation from [1] included a sigmoid activation function on the logits. Empirically, however, we found that softmax activation produces better accuracies. This is likely because softmax adds a soft constraint of modelling all the attributes in a class as a probability distribution, whereas sigmoid treats every attribute as a binary classification. Since the images are such that each class only has one attribute, there are some dependencies between attributes in the same class. Hence, this constraint provided by softmax may be more favourable.

Altogether, we have a pretrained VAN-T feature extractor backbone, followed by a dense feedforward network, trained with an asymmetric loss. Our model has a total of 4,174,426 trainable parameters. We opted for Adam as our optimizer, with initial learning rate of 5×10^{-4} and weight decay of 3×10^{-4}. We also used a cosine-annealing learning rate scheduler to reduce our learning rate progressively, setting `T_max` to be the total number of training epochs, which we set at 100.

Before we train the model with our training images, we must perform image preprocessing. Because we are using a pretrained VAN-T feature extractor backbone, we follow an identical preprocessing pipeline as the pretraining, which includes resizing the images and normalizing them. Prior to the preprocessing, we included some other transformations such as random horizontal flipping, rotation, and colour jittering to reduce overfitting. We applied these transformations and preprocessing on all the training images. Additionally, to address the imbalance dataset issue, we applied mix-up image augmentation on all the minority images. This makes an interpolation between minority images and some other training images, including their labels. Mathematically, give some minority image, x_m, with label y_m and some training image, x_t, and label y_t, we have

$$\begin{aligned} x'_m &= (1-\lambda)x_m + \lambda x_t \\ y'_m &= (1-\lambda)y_m + \lambda y_t, \end{aligned}$$

where $\lambda \sim \mathrm{Beta}(\alpha, \beta)$. Then, we use x'_m and y'_m as our training image and label, in place of x_m, and y_m. Effectively, the model would be exposed to different minority images during each epoch, since x'_m varies with the choice of x_t and λ, allowing the model to better generalize and classify these rare images. We define minority images to be those with an attribute whose count is less than a `minority_threshold`, and we take $\alpha = 0.2 = \beta$.

Finally, because the VAN-T feature extractor is relatively light-weight, we could fine-tune the entire feature extractor, together with our classifier head on our model, for a potentially better performance. We trained the model for a total of 100 epochs on one RTX 4060 Ti (16 GB) GPU, taking roughly 96 minutes.

3. Results and Discussions

For brevity, we will call the average per-class accuracy as accuracy. Figures 2 and 3 below show the training and validation losses in the same axes, and training and validation accuracies in the same axes, during the 100 epochs of training, respectively.

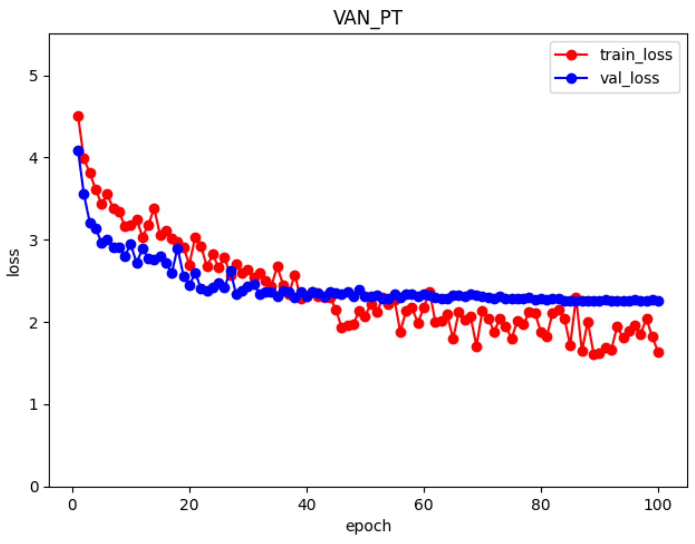

Figure 2: Training (red) and validation (blue) losses during training.

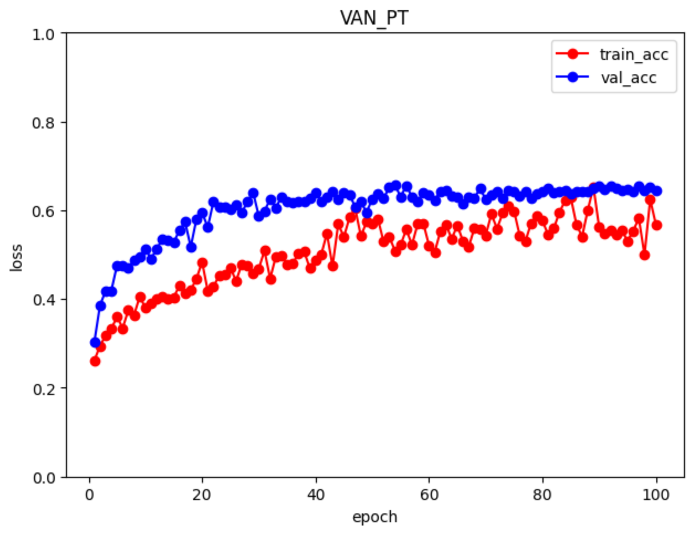

Figure 3: Training (red) and validation (blue) accuracies during training.

We can see that both the training and validation losses seem to steadily decrease, while the training accuracies and validation accuracies increase, over the 100 epochs. This suggests that the model is learning to better classify the images, even for unseen images in the validation set. Although the training loss eventually became lower than the validation loss, the validation accuracy was consistently higher than the training accuracy. This is due to the transformations and augmentations performed on the training images, resulting in noisier images that the model must use to classify on, hence a poorer result. Along the same vein, notice that the trends of the validation loss and accuracy are much smoother than those for the training set, since the training images are slightly different for every epoch, due to the transformations and augmentations.

As we aim to achieve the highest test accuracy, we saved two checkpoints of our model training: The first is the weights corresponding to the lowest validation loss, while the second is the weights corresponding the highest validation accuracy. In this case, the latter achieved a better test score of 0.62883, while the former got 0.6130. This means that the model with the highest validation accuracy might have struck a better balance between correctly classifying the minority classes (due to ASL), while not neglecting the majority class completely.

Besides this configuration, we also experimented with other methods such as performing mix up between all the training images, performing mix up solely between minority images, implementing dice loss and implementing weighted cross entropy loss to address the imbalance dataset issue. Ultimately, however, the abovementioned configuration produced the best results by better handling the imbalanced data, either through contribution to loss function or generalizing minority images. Surprisingly, performing mix up solely between minority images did not improve the performance. One possibility is that the interpolated images may lie too far out of the underlying distribution, since both images used are deemed as minority (i.e., very unlikely to observe in reality). Moving forward, however, we could explore other techniques to address the imbalance dataset issue. For instance, we could apply oversampling on the minority images so as to expose these otherwise rare images to the model more often.

4. Conclusion

In this multi-label classification task, with an imbalanced dataset, we managed to train a model comprising of two parts: a feature extractor backbone from VAN-T pretrained on ImageNet-1000, and a custom feedforward classifier network consisting of two hidden layers. Despite a lightweight model, with slightly less than 4.2M parameters, we obtained an average per-class accuracy of 0.62883 on the test set, by using an asymmetric loss and image augmentation techniques to address the issue of having an imbalanced dataset.

5. References

[1] Alibaba-MIIL. (n.d.). Loss functions for ASL. In ASL/src/ loss_functions/losses.py. Retrieved March 28, 2024, from https://github.com/Alibaba-MIIL/ASL

[2] Ben-Baruch, E., Ridnik, T., Zamir, N., Noy, A., Friedman, I., Protter, M., & Zelnik-Manor, L. (2021). Asymmetric Loss for Multi-Label Classification. https://arxiv.org/pdf/2009.14119.pdf.

[3] Guo, M.-H., Lu, C.-Z., Liu Z.-N., & Hu, S.-M. (2015). Visual Attention Network. *Journal of LaTeX Class Files,* 14(8), 1. https://arxiv.org/pdf/2202.09741.pdf.

[4] Visual Attention Network. (n.d.). Retrieved March 28, 2024, URL: mmpretrain/configs/van at main · open-mmlab/mmpretrain (github.com).