Multi-Label Classification

Jiaming Lin
jlin0701@uni.sydney.edu.au
470345744

Ziying Huang
zhua4138@uni.sydney.edu.au
480359623

Lehan Yang
lyan3310@uni.sydney.edu.au
500136245

1 Introduction

Since the success of AlexNet [1] in the 2012 ImageNet challenge, the deep learning model has become a pervasive choice for classification problems in various domains. Among all the tasks that are tackled, single-label classification tasks tend to be the focus, whereas instances, particularly images in the real-world are inherently multi-label [2].

Multi-label classification (MLC) problems are important. Compared to the single-label classification counterpart, it aligns better with the real-world image classification tasks, and hence the investigation is more practically demanding. Beyond that, given the gradual saturation of single-label classification models, more attention to MLC problems may unlock more breakthroughs that may possibly benefit the deep learning community as a whole.

In this study, our aim is to investigate the most high-performance method to conduct multi-label classification over the given image-caption dataset. Firstly, an exploratory analysis of the dataset is conducted, through which we identify a severe class imbalance issue. To alleviate this, we adopt more appropriate train-val sampling method to balance the class split and utilize the sample-F1 metric that are more robust to class imbalance issue. Given that the provided dataset has two modalities and combining complementary information from various modalities tends to improve the performance, we investigate the potential multi-modal model to combine the image and text information effectively. To achieve this, we first identify the best image model by experimenting with a collection of high-performance CNNs and transformer variants in the literature. Swin-L and Swin-V2-L, as the two most competitive image models, are selected to combine with the BERT model using two candidate multi-modal combination approaches for the final prediction. Besides choosing the most appropriate models, we also experiment with a variety of tricks, hoping to further push the performance forward. This includes pre-training and fine-tuning image models on the larger ImageNet-21K and ImageNet-1K datasets, implementing novel data augmentation methods, such as mixup and test-time augmentation in the training and testing stages, respectively. Ultimately, our final best model achieves a sample F1 score of 90.79%.

2 Related Work

2.1 Image Classification Models

2.1.1 Convolutional Neural Networks (CNNs)

There has been increased research on different convolutional neural network (CNN) architectures over decades. Many of them achieved impressive performance on image recognition tasks. In this section, we discuss three commonly used CNNs, including RestNet, DenseNet, and EfficientNet.

The ResNet architecture was developed in 2016 by He et al. [3]. Residual blocks were introduced in the paper, and each residual block contains a few layers and a shortcut connection. The shortcut connection was used to skip the unuseful layers for easier optimization and to avoid the vanish gradient by simply stacking layers.

The DenseNet architecture [4] also utilized shortcut connections, like ResNet [5]. However, to maximize information flow, each layer in DenseNet has direct connections to all other layers. For example, the l^{th} layer would receive feature maps from the entire previous layer from layer 0^{th} to $l-1^{th}$ as input:

$$x_l = H_l\left([x_0, x_1, ..., x_{l-1}]\right) \tag{1}$$

where H_l denotes a composite function of a batch normalization, a rectified linear unit, and a 3x3 convolution; x_0 to x_{l-1} denotes a concatenation of feature maps from previous layers [5].

EfficientNet [6] introduced a new way to scale up a ConvNet [7]. Before it was introduced, the accuracy increase will quickly become saturated when arbitrarily scaling up a ConvNet along a single dimension. EfficientNet balanced 3 dimensions: width, depth, and resolution by uniformly scaling them with a constant ratio. With the compound scaling method, EfficientNet achieved higher accuracy and a huge reduction in FLOPS and model size on ranges of public image datasets.

2.1.2 Vision Transformers

The transformer model [8] was initially developed for machine translation tasks, where a word sequence in the source language is required to be translated into another sequence in the target language. To tackle such tasks, the proposed architecture, as illustrated in Figure 1, consists mainly of an encoder (middle row) and a decoder (bottom row). Specifically, the encoder takes the input word sequence, converts it to an input embedding, and then passes it to the encoder blocks to produce the encoded input sequence. The transformer decoder then takes the encoded sequence as well as the previously predicted output sequence to predict the next word in the output sequence. Both encoder and decoder contain repeated blocks from which multi-head attention (right module in the first top row) as well as feed-forward layers lie. Multi-head attention comprises multiple self-attention blocks, helping to capture several complex relationships among all words of a sequence.

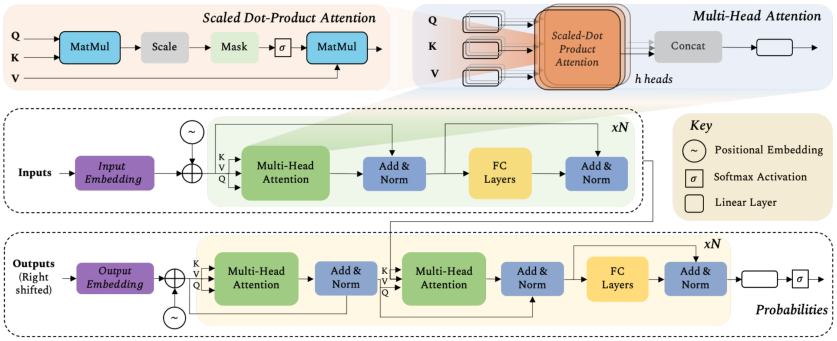


Figure 1: The architecture of transformer model [9]

In summary, transformer models provide multiple distinguishing advantages compared to the CNN and RNN counterparts. First, the attention mechanism attends a complete sequence, which enables transformers to capture long-term dependency between words that LSTM is incapable of. Second, unlike CNNs where inductive bias (i.e. prior knowledge) is posed, transformers require minimal prior knowledge of the data and therefore the model capability is less when larger-scale data is available [10]. Finally, the transformer design allows multiple modalities to be easily processed using similar processing blocks and can be conveniently scaled to very large networks [9].

Motivated by these benefits, in the computer vision community, much progress has been made in adapting the transformers to vision tasks. Vision Transformer (ViT) [11] is the first work that shows how pure language transformers can possibly replace standard CNNs on *large-scale* image datasets. To achieve this, the authors directly apply the standard language transformer architecture [8] pre-trained on a large-scale image dataset to a sequence of image patches similar to words. Consequently, the models match or exceed the performance of SoTA CNNs while being relatively inexpensive to pre-train. However, such success is highly dependent on the dataset size model pre-trained over. ViT is pre-trained over a 300 million image dataset internal to Google, limiting their adoption to the general public. Fortunately, the data-efficient version of ViT, namely DeiT [12], is proposed soon enough after the ViT proposal. By training on a much smaller ImageNet1K, DeiT demonstrates that high performance transformers can also be learned on mid-size image datasets. The main contribution of DeiT is the proposal of a novel transformer-specific distillation strategy, which exploits a stronger CNN teacher model to train the transformer student model. Concretely, the distillation procedure is illustrated in Figure 2. Image patches, also referred to as patch tokens in the paper, are first processed by the normal transformer layers, and then in the end output the predicted class token and distilled token. The normal cross-entropy loss is calculated between the predicted class token and the true class token, which is then minimized to enforce correct predictions. Meanwhile, a new distillation loss between the predicted distillation token and the CNN teacher model output is also simultaneously minimized to encourage the distillation token to match the teacher's prediction. Beyond distillation, the authors also applied common data augmentation techniques for CNNs to DeiT to further alleviate the data-insufficiency issue.

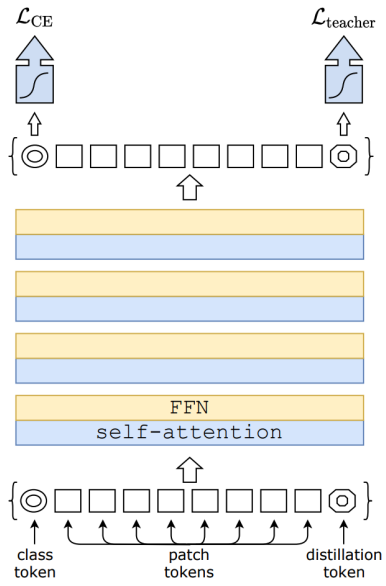
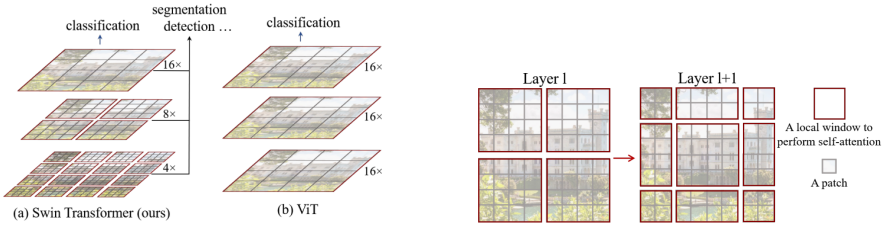


Figure 2: Distillation procedure of DeiT

However, both the ViT and DeiT transformers still fail to address two key challenges specific to vision tasks. The first is related to the scale. Visual elements tend to have larger scale variance compared to text counterparts, and therefore a fixed token design (e.g. 16 fixed tokens of ViT in Figure 3(a)) is limited in capturing more granular features in each token. Another challenge is related to computational complexity. Due to the global self-attention mechanism, the feature maps of ViT and

DeiT scale quadratically with the input image size. Therefore, both models are limited in their application to more general vision tasks, including dense prediction tasks that require more fine-grained feature extractions. The proposal of a Swin Transformer [13] overcomes both issues. The hierarchical feature map design in Figure 3(a) not only allows more fine-grained features to be extracted, but also ensures linear complexity to the input image size by restricting self-attention to each local window. Beyond that, the shifted window design in Figure 3(b) promotes the cross-window connection, capturing global contextual information to further enhance the representation power. As a result, Swin outperforms many SoTAs in various vision tasks, demonstrating its potential as a more general-purpose backbone for computer vision [13].



(a) Hierarchical feature maps (b) Shifted windows

Figure 3: The two key ingredients in Swin Transformer [13]

However, two challenges remain in the vision community. The first challenge is the scaling up of vision models has been lagging behind, and the other challenge is that a resolution gaps exist between pre-training and fine-tuning. Swin Transformer V2 [14] is proposed to address both issues by making several adaptations to the original Swin Transformer architecture [13]. Firstly, a res-post-norm is used to replace the previous prenorm configuration. Second, a scaled cosine attention is provided to replace the original dot product attention. Finally, a log-spaced continuous relative position bias approach is adopted to replace the previous parameterized approach. The first two adaptions aim at scaling up the model capacity, whereas the last adaption aims at scaling up the resolution. The performance uplift of these adaptions is astonishing. Swin Transformer V2 sets new performance records on four representative vision benchmarks, further closing the capacity gap between vision and language models.

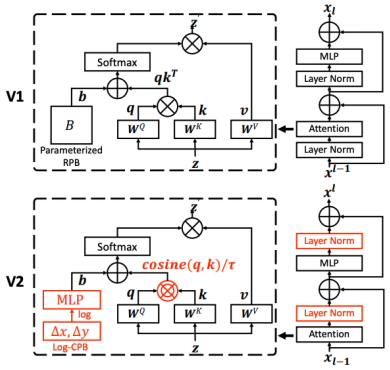


Figure 4: The adapted architecture of Swin Transformer V2

2.2 Language Models

The text captions provided with each image could be useful in combination with the image model to improve performance. The image model discussed in the previous section has not utilized this information. In this section, we investigate a potential approach on captions, Bidirectional Encoder Representations from Transformers (BERT).

2.2.1 BERT

BERT [15] is a transformer-based architecture, which was first introduced by Google in 2017. This architecture adopts a self-attention mechanism which is used mainly in the field of natural language processing (NLP). The traditional transformer proposed in the original paper [15] has 6 encoder blocks composed of feed-forward and attention layers, responsible for text input and processing. It also has 6 decoder blocks to produce predictions. This architecture enables the sequence input to be read in one direction. BERT, as an advanced version of the transformer, was introduced to use bidirectional training, taking into account the previous and next tokens, with the aim of enriching the language context.

The BERT architecture consists of encoders only. The BERT-base introduced in 2019 [16] consists of 12 encoder blocks, 12 attention heads, and 110 million parameters. A sequence of tokens was given to the first encoder. The final input to the first encoder consists of a token embedding, a sequence embedding, and a position embedding, shown in Figure 5.

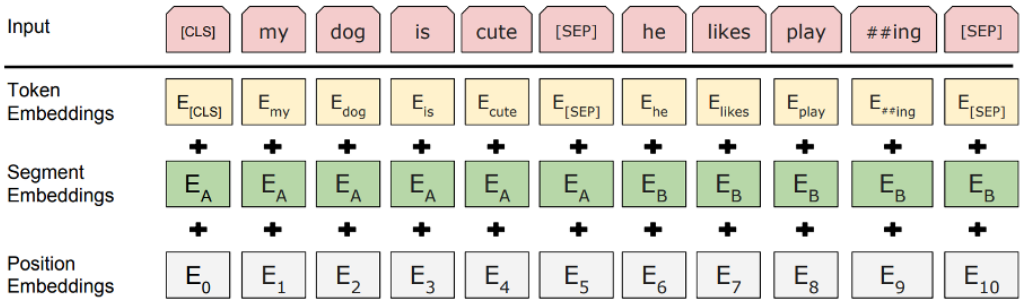


Figure 5: BERT input summation [16, Figure 2]

The token embedding has [CLS] and [SEP] as the special token, indicating the beginning of the first sentence and the end of each sentence, respectively; segment embedding adds the segment to distinguish between sentences; and the position embedding gives the information of the position of each token.

The sequence input to BERT travels up the encoder stacks. Like in each traditional transformer block, the input in BERT will first pass through a self-attention layer, followed by a feed-forward neural network, and then to the next block. Two methods in BERT, Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) methods, were used to achieve bidirectional training. MLM was carried out by randomly masking 15% of the input and trained to predict the masked input using the remaining as a context. NSP worked to understand the relations between sentences. In training, sentences would be paired and then given a binary prediction of whether the second sentence in the pair is the next sentence of the first one.

3 Methods

3.1 Exploratory Analysis

In order to determine the most suitable methodology for the given multi-label classification task, we first conduct an exploratory analysis of the provided dataset.

Overall, the dataset contains 40K samples where 30K are training samples and the remaining 10K are testing samples. Each training sample is associated with an image, a descriptive caption as well as corresponding label(s) ranging from 0 to 19. Samples in the testing set follow a similar structure, except the label(s) are hidden for testing purpose.

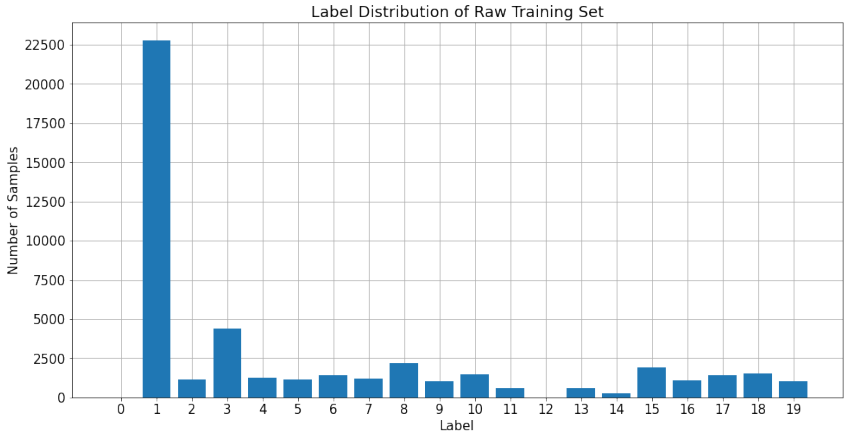


Figure 6: Label distribution in the raw training set

Figure 6 depicts the label distribution of the provided 30K training dataset, which is clearly unbalanced. Among all 20 labels (0 to 19), label 1 completely dominates the distribution while label 0 and 12 are absent from all training samples. Besides these labels, the remaining all have similar numbers of occurrence (less than 5000 samples). This is clearly an issue we need to account for when designing the methodology, and we propose to address this in the following two ways:

1. *Stratified Sampling* We split the training and validation data in a stratified way such that the labeling distribution is well balanced between the splits. Specifically, we utilise the iterative_train_test_split module in the skit-multilearn library, which is designed specifically for the multi-label data.
2. *Robust Metrics* Metrics such as macro-F1 are known to be vulnerable when it comes to imbalanced labels [17], therefore it is crucial to select the appropriate metrics given the imbalance issue. As disclosed in Ed, leaderboard evaluation adopts sample-level F1 measure, which should be naturally robust to the class-imbalance issue, since it does not evaluate class-wise. Therefore, we chose sample F1 as our final robust metric.

3.2 Image Models

3.2.1 Model Architecture

CNNs Convolutional Neural Networks have been the pervasive choices in deep learning for almost a decade, and a diverse collection of architectures have been proposed. To be inclusive, we select the two widely-adopted models, ResNet [3] and DenseNet [4], along with a more recently developed but smaller-scale model EfficientNet [6] as our baseline models. The relevant principles can be referred to in Section 2.1.1.

Vision Transformers As mentioned in the related work section, vision transformers offer several unique advantages that CNN cannot provide otherwise. Among this group, three transformer models, Deit [12], Swin [13] and Swin-V2 [14] are selected, where the relevant principles can be referred to in Section 2.1.2. These models are chosen for the reasons that:

1. DeiT, Swin, and Swin-V2 are highly ranked on the ImageNet benchmark, which is the most commonly used benchmark for image classification tasks. Given that, we believe that such high performance is likely to generalize to our classification task.
2. In terms of performance, the Swin transformer and its later adaption Swin-V2 [14] are the most impressive ones given they outperforms many SoTAs in various vision tasks performance (see Section 2.1.2).
3. In the perspective of features, both hierarchical feature maps and shifted window designs of Swin help to capture more granular features in the images, further enhancing the power of extracted representations. Intuitively speaking, these designs should also help to capture more fine-grained multi-label features existing in each image.

For both DeiT and Swin, we implement all architecture variants. After empirically identifying Swin-L as the best Swin model, we then implement the adapted Swin-L, namely, Swin-V2-L and see if the adaption can bring further performance improvement.

Extra Linear Layer with Sigmoid Given the above selected architectures, we further adapt them by adding an extra linear layer in the end. There are two associated reasons:

1. It helps to map the output feature dimension of the pre-trained model (i.e. 1000, more details described in Section 3.2.2) to the desirable one for our task (i.e. 20).
2. It takes into account the cooccurrence of labels, which is common in multi-label classification tasks [2]. Applying a linear layer W to the output features of the pre-trained model X, we can obtain a linear combination of features. Intuitively, this helps capture the feature co-occurrence, hence label co-occurrence, boosting the label prediction performance. An example could be that skateboard and man are often co-existing in a single image.

The output of the linear layer is then passed through a sigmoid function to output probabilities for all labels. Labels with probability larger than the threshold will be output as predicted labels. The threshold will be tuned for each model in Section 4.1.

3.2.2 Pretraining and Finetuning

We employ the conventional 'pre-training and fine-tuning paradigm in the vision community and pre-train all image models on the larger ImageNet dataset. Although it is believed that pre-training on ImageNet-21K could provide better performance for large models [18], the pre-training model does not always seem to be available. This is the case for DeiT, as to the best of our search, the ImageNet-22K pre-trained model seems to be not available. Therefore, we chose to pre-train the DeiT model using ImageNet-1K instead and then fine-tune it over our given dataset. For all Swin and Swin-V2 variants, we pre-train them first on ImageNet-22K, fine-tune them on ImageNet-1K, and finally fine-tune them over our given dataset. The tuning detail can be found in Section 4.1.

Intuitively, the models should be able to learn a transferable feature representation as ImageNet is not only an image dataset but also covers way more images and classes. Once features are learned, we add a linear layer after the extracted feature layer, mapping the raw feature dimension (i.e., 1000 classes) to the desirable dimension (i.e., 20 classes).

3.2.3 Image Augmentations

Random Augmentation For the given image data set, we choose the RandAugment data augmentation method [19] to artificially increase the size of the training data. We believe it is necessary as we adopt transformer models in our experiment, which are known to be more 'data-hungry' than the CNNs. In addition, the DeiT paper [12] empirically confirms that augmentation can bring further performance gains to the general transformer models.

Mixup Augmentation Deep neural networks are powerful, but they can also be prone to memorising noisy corrupted samples due to its outstanding expressiveness. This could be a relevant problem in our study. In the data exploratory stage, we found that label 1 tends to be highly correlated with man. However, multiple images with label 1 do not have any man contained in them and it could be to do with mislabelling. To take this potential issue into consideration, we apply mixup [20], a data-agnostic augmentation technique, to the training data and construct artificial training examples through a convex combination of existing ones. This encourages the trained model to have a more linear behavior in-between training examples and therefore to be less overconfident towards the predictions.

$$\begin{aligned} \hat{x} &= \lambda x_i + (1-\lambda)x_j, \\ \hat{y} &= \lambda y_i + (1-\lambda)y_j, \end{aligned}$$

where $\lambda \in [0, 1]$ is a random number

Figure 7: Mixup augmentation

Test-Time Augmentation Test-time augmentation, also known as TTA, is another augmentation technique that applies only to the test set. Specifically, it first creates multiple augmented copies for each test image, and then aggregates the model predictions over all copies to produce a final prediction. Intuitively, by averaging multiple predictions and smoothing out the loss, TTA should be able to correct some overconfident wrong predictions and therefore possibly improves performance [21].

3.3 Language Models

Captions went through a simple pre-processing step for our language model BERT. These included removing punctuation, correcting symbol errors, removing trailing whitespace, and removing entity mentions. With the novel language model BERT, with the advantage of using the novel masked language modeling, it can learn the contextual information of words, giving it a greater possibility of success in a text classification task. The relevant principles can be referred to in Section 2.2.1.

Figure 8 shows an overview of our BERT model.

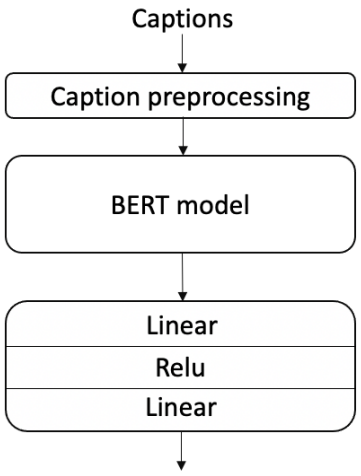


Figure 8: An overview of our BERT language model.

3.4 Multi-Modal Models

Combining complementary information from various modalities was found to improve the performance of the model [22]. This idea was tested in our study. We utilized information from both texts and images by combining our best language model and the image model. In this section, two approaches have been investigated for multi-modal learning.

One approach was to use a concatenate layer. Provided the ability to join the output of the language model and the image model. The concatenation approach can be found in the Pytorch framework with torch.cat(). The second approach was to use the bilinear layer. It can also effectively combine multimodal features by multiplying the output of two models using the outer product, pooling to obtain a bilinear vector, and then passing through the output layer to obtain predictions [23]. The method can be found in Pytorch framework using torch.nn.Bilinear().

3.4.1 Concatenation approach

As shown in Figure 9, both the image model and the language model were concatenated using a concatenate layer and then passed through a linear layer for the purpose of reducing dimensions, to the same dimensions as the number of classes. And then passed through a sigmoid output layer. Let x^i, x^c be the output of the image and caption models, and W be the weight matrix:

$$output^{concatenation} = \sigma\left(W^{40\times 20}\left[x^i; x^c\right]\right) \tag{2}$$

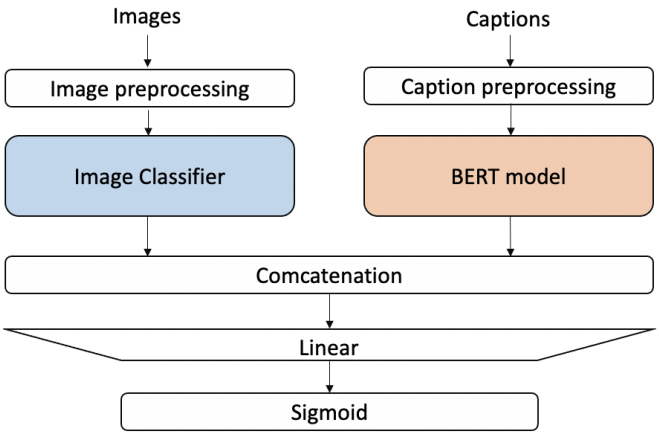


Figure 9: An overview of our concatenation approach for combining the output of the models.

3.4.2 Bilinear approach

For our bilinear approach shown in Figure 10, both the image model and the language model were passed through a bilinear layer and then immediately output through a sigmoid layer. Let x^i, x^c be the output of the image and caption models, and W be the weight matrix:

$$output^{bilinear} = \sigma\left(x^i W^{20\times 20\times 20} x^c\right) \tag{3}$$

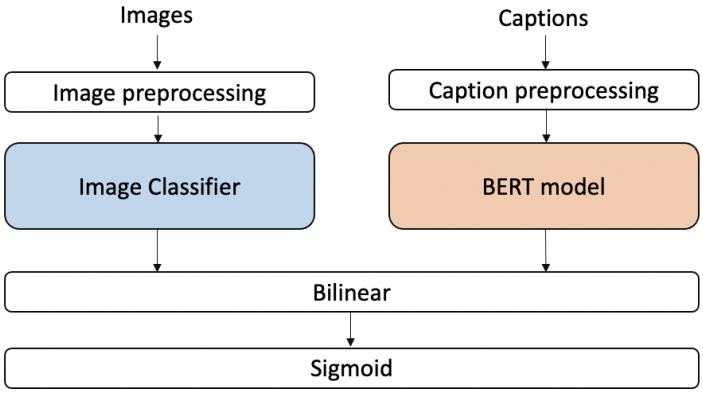


Figure 10: An overview of our bilinear approach for combining the output of the models.

4 Experiments

4.1 Parameter Tuning and Model Selection

Sigmoid Threshold We tune the threshold of sigmoid output probability, which determines if each candidate label will be included in the final predicted label set. It is necessary to find an appropriate threshold. Intuitively, a threshold that is too high can lead to much fewer labels predicted than the actual number of correct labels. Meanwhile, a threshold that is too low can output way too many incorrect labels. We choose three candidate thresholds which are evenly spaced around 0.5. More extreme values outside the range are not tuned since they are less reasonable. CNN-EfficientNet is the best performing CNN model out of all three candidate CNNs. Given that it serves as our baseline and its performance hierarchy with other image models is quite clear given the same threshold, we choose to not tune extra thresholds.

Multimodal Approach For image-only model group, both Swin-V2-L and Swin-L are high-performing, with performance much higher compared to the remaining models. For both of them, we attempt multi-modal models by including the BERT model and tune for the two candidate multi-modal approaches being applied.

Input	**Model**	**Threshold**		
		0.4	0.5	0.6
Image	CNN-EfficientNet	N/A	83.62%	N/A
	DeiT-S	86.02%	**86.06**%	84.53%
	Swin-L	89.73%	**90.07**%	89.60%
	Swin-V2-L	90.71%	**90.79**%	90.58%
Caption	BERT	**87.08**%	86.79%	86.37%
Image + Caption	Swin-L + BERT (Bilinear)	**90.03**%	89.97%	89.75%
	Swin-L + BERT (Concatenate)	89.91%	**89.99**%	89.65%
	Swin-V2-L + BERT (Bilinear)	**90.17**%	90.17%	89.94%
	Swin-V2-L + BERT (Concatenate)	90.06%	**90.27**%	90.09%

Table 1: Table for threshold and multi-modal approach selection

4.2 Model Result Summary

In this section, we present the highest performance model for each model group after tuning the threshold. Our best performing model is "Swin-V2-L", with validation sample-F1 being 90.79%. This differs from our original expectation as it outperforms the best multi-modal model with the same image model component, meaning that the extra information provided by caption is not effectively utilized. We suspect it could be to do with how mixup interacts with the caption data, and this is detailed in the discussion section.

Input	**Model**	**Sample F1**
Image	CNN-EfficientNet	83.62%
	DeiT-S	86.06%
	Swin-L	90.07%
	Swin-V2-L	**90.79**%
Caption	BERT	87.08%
Image + Caption	Swin-V2-L + BERT (Concatenate)	90.27%

Table 2: Summary of model validation sample-F1

4.3 Ablation Study: Best Model

Model	Sample F1
Swin-V2-L without Mixup	88.56%
Swin-V2-L	90.79%

Table 3: Ablation Study of the Best Model

5 Conclusion and Discussion

We investigate the most appropriate multi-label classification model by extensively experimenting with representative image models, multi-model combination approaches, along with a variety of performance tricks. We conclude Swin-V2-L as our best final model with the validation sample-F1 being 90.79%.

The best multi-modal model, Swin-V2-L + BERT (concatenate), does not produce the expected outcome during our limited experiments. Both multi-modal approaches, concatenation and bilinear, lead to a decrease in overall performance. We suspect that it could be related to the improper deployment of MixUp augmentation. In this study, we did not manage to investigate the issue properly given the long model running process as well as our limited computational resources. In future work, we hope to investigate more empirically and possibly experiment with a more NLP-adapted mixing method.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[2] Weiwei Liu, Haobo Wang, Xiaobo Shen, and Ivor Tsang. The emerging trends of multi-label learning. *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[4] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[5] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269. IEEE, 2017.

[6] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[7] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[9] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM Computing Surveys (CSUR)*, 2021.

[10] Stéphane d'Ascoli, Hugo Touvron, Matthew L Leavitt, Ari S Morcos, Giulio Biroli, and Levent Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. In *International Conference on Machine Learning*, pages 2286–2296. PMLR, 2021.

[11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[12] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[13] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.

[14] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. Swin transformer v2: Scaling up capacity and resolution. *arXiv preprint arXiv:2111.09883*, 2021.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[17] Sklearn Developer. Sklearn documentation. Available at `https://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-f-measure-metrics`.

[18] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *European conference on computer vision*, pages 491–507. Springer, 2020.

[19] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020.

[20] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[21] Nathan Hubens. Test time augmentation (tta) and how to perform it with keras. *Medium*, 2021.

[22] Kuan Liu, Yanen Li, N. Xu, and P. Natarajan. Learn to combine modalities in multimodal deep learning. *ArXiv*, abs/1805.11730, 2018.

[23] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear cnn models for fine-grained visual recognition. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1449–1457. IEEE, 2015.

6 Appendix

6.1 Environment

6.1.1 Hardware

CPU: Intel Xeon Gold 6130 (16) @ 3.70 GHz (AVX-512) * 2
Memory: 128 GB
GPU: Nvidia RTX A4000 * 2

6.1.2 Software

System: Ubuntu 20.04

Python: 3.9.7

Packages:

```
nltk
numpy
pandas
torch
transformers
sklearn
PIL
wandb
randaugment
tqdm
timm
```

6.2 Code instructions

6.2.1 Data Preparation

Before train the model, we need first unzip the dataset and put them in the original order into ”data” folder.

6.2.2 Training with Customize Parameters

We can customize the parameters to training the model for the multi label classification task. We have separated the text classification part, the image classification and the multi modal part.

To train a BERT:

```
python bert_train.py
```

To do inference on test set using bert, run:

```
python bert_test.py
```

It will show the metrics every epoch and save the best model to ”./bert_best.pth”

And to train a image only model, you can specify the model, learning rate, epoches, lr scheduler and etc by: Below listed all the modifiable arguments that user can edit, just give a brief view of the whole parameters.

```
python train.py --exp_name training \
                --epochs 20 \
                --data_dir data \
                --batch_size 32 \
                --img_size 224 \
```

```
                --lr 1e-5 \
                --step_size 8 \
                --gamma 0.5 \
                --seed 3407 \
                --model swin_large_patch4_window7_224_linear \
                --weighted_loss False \
                --mixup True \
                --mixup_alpha 0.4 \
                --autoaug False \
                --focal_loss False \
                --freeze_epochs 0 \
                --after_freeze_lr 1e-5 \
                --test swin_large.pth \
                --load swin_large.pth \
                --load_combine combine.pth \
                --tta True \
                --label_smoothing 0 \
                --kd_alpha 0 \
                --kd_temp 2 \
                --kd_teacher swin_teacher_large.pth \
                --bilinear_text True \
                --concat_text False \
                --threshold 0.5
```

6.3 Kaggle Submission

At this section, we listed the 2 submissions we finally chose for private leaderboard, as well as their training, testing and ensemble procedure.

6.3.1 Submission 1

```
Ensemble of Swin-Transformer V2 Large 256 and Swin-Transformer Large 224
LB 0.90125
SIZE: 756M + 752M = 1508M
TOTAL TIME COST: 5 Hours (Two cards)
```

To train this Swin-Transformer Large 224:

```
python main.py --exp_name swin_large_224 \
               --model swin_large_patch4_window7_224_linear \
               --step_size 8 \
               --gamma 0.5 \
               --epochs 20 \
               --lr 0.00001 \
               --mixup True \
               --img_size 224
```

To train this Swin-Transformer V2 Large 256:

```
python main.py --exp_name swinv2_large_256 \
               --model swinv2_large_window12to16_192to256_22kft1k \
               --step_size 8 \
               --gamma 0.5 \
               --epochs 20 \
               --lr 0.00001 \
               --mixup True \
               --img_size 256 \
               --batch_size 8
```

Then we can got two trained and saved model in checkpoint folder:

```
swin_large_224_best.pth
swinv2_large_256_best.pth
```

To do the inference on the testset and get submission csv:

```
python main.py --test swin_large_224_best.pth \
               --model swin_large_patch4_window7_224_linear
python main.py --test swinv2_large_256_best.pth \
               --model swinv2_large_window12to16_192to256_22kft1k
```

Then we can get two submission csv in the submissions folder, open ensemble.ipynb, type in each submission's leaderboard score as weight, do the ensemble and get the final submission csv.

6.3.2 Submission 2

```
Swin-Transformer V2 256
LB 0.90090
SIZE: 756M
TOTAL TIME COST: 5 Hours (single card)
```

To train this model:

```
python main.py --exp_name swinv2_large_256 \
               --model swinv2_large_window12to16_192to256_22kft1k \
               --step_size 8 \
               --gamma 0.5 \
               --epochs 20 \
               --lr 0.00001 \
               --mixup True \
               --img_size 256 \
               --batch_size 8
```

To do the inference on the testset and get submission csv:

```
python main.py --test swinv2_large_256_best.pth \
               --model swinv2_large_window12to16_192to256_22kft1k
```