THE UNIVERSITY OF SYDNEY

The University of Sydney

School of Computer Science

COMP5329 Assignment 2

Author:
Zhen Yang 530549354
Mingyue Ma 530400789
Yixin Pei 540093544

Supervisor:
Chang Xu

An Assignment report submitted for the COMP5329:

COMP5329

May 17, 2024

Contents

1 Abstract

This report investigates methods for enhancing performance in multi-label image multi-classification problems. Subsequent sections will cover the problem introduction, analysis, dataset details, model overview, experimental outcomes, and a summary with reflections.
The link to the code and data is link.

2 Introduction

2.1 Aim of Study

In deep learning, multi-label classification plays one of the most important roles in complex tasks such as recognition and automatic tagging, as most images in real life may contain more than one attribute and labels. The aim of this study is to build a deep learning model which can make predictions to the multilabel of images with a great enough accuracy and limited computing resources. In order to find the most suitable model, three types of models are used in the experiments of this assignment, which are image-based model, text-based model and a more advanced model which can use the image data and text data at the same time. Considering each sample in the dataset contains an image, a multilabel and a caption of the image, this experimentation can be used to help find out the best method that can be applied to this study.

In multi-label classification tasks, each sample can have more than one label so that the traditional classification method is not suitable for this type of tasks. Also, the data collected from the real world are very likely to have class imbalances. To address these challenges, this study will compare the performance of different models, conducting hyperparameter tuning and ablation studies. Through the experiments and comparisons in the F1-score of different models, the method which can interpret and utilize the most of the visual and textual data to improve the accuracy and efficiency of the multi-label classification task will be found.

2.2 Important of Study

The importance of this study is advancing AI applications. As the development of technology, there is a growing demand for more sophisticated deep learning methods. This study's outcomes have better performance than traditional methods. And it could influence other AI fields that combine visual and textual data. This integration of text and vision models could change how these systems understand and interact with the environments. Moreover, the importance of this kind of deep learning model is also across various sectors. For instance, in the healthcare sector, it can improve diagnostic accuracy by providing deeper insights from medical images. In sectors like autonomous driving and agriculture, it enhances vehicle safety by better interpreting road scenes and textual cues, and enables precise crop monitoring and optimized resource use. Additionally, as students, we gain more practical experience with current

models. This study will enhance our understanding of how to apply materials from textbooks to real world problems. And it can contribute to our future professional path.

2.3 Introduction of Method and Motivation

The methodology applied in this assignment involves comparing three types of models: an image-based model, a text-based model, and a multimodal model that combines the characteristics of both. The motivation behind this approach is to improve the performance of the model while keeping it compact enough to be applied to devices with limited computing resources. This is particularly relevant for our task, which involves multi-label image classification with captions.Given that each sample in our dataset includes an image, multiple labels, and a caption, it is essential to leverage both visual and textual information to enhance classification accuracy.

For image data, we use Convolutional Neural Networks (CNNs), which are effective in extracting visual features. For text data, we apply Natural Language Processing (NLP) techniques to interpret the captions. By exploring these models individually and in combination, we aim to identify the approach that yields the best performance in terms of accuracy, model size, and training time.The motivation for using this combination of models is to ensure that we can accurately classify images with multiple labels while maintaining a lightweight model suitable for deployment on devices with limited computational power. Hyperparameter tuning will be performed on the best-performing model to further optimize its performance.

In summary, our goal is to find a model that offers the highest accuracy while balancing size and computational efficiency. This involves analyzing image-based, text-based, and multimodal models to determine which approach is most effective for our specific task, ultimately leading to a solution that is both accurate and efficient.

3 Related works

3.1 Multi Label Image Classification

Multiple architectures have been researched for multi-label image classification, with CNN being the most popular due to its excellent performance in image-related tasks (“Conference on Computer Vision and Pattern Recognition”, 2009). CNN’s capability in multi-label image tasks was initially demonstrated in 2014 (“CNN: Single-label to Multi-label”, 2014). Then, recognizing the need for a relatively large dataset to train efficient CNN models for multi-label tasks, pre-trained CNN-based models have become popular, with VGG and ResNet being representative architectures and reference points.

VGG introduced multiple 3x3 convolution layers to replace larger-sized convolution layers such as 5x5 and 7x7 (“Very deep convolutional networks for large-scale image recognition”,

2014). This deepened the network's depth, thereby enhancing model performance to some extent. Additionally, due to the smaller number of parameters in 3x3 convolution layers compared to larger ones, this substitution reduced the overall number of parameters in the network, consequently reducing the model's size ("Very deep convolutional networks for large-scale image recognition", 2014). VGG16 and VGG19 (Pal, 2016) are commonly used versions for evolution and real-world application, as well as for model comparison. For example, Nath, Chaspari, and Behzadan utilized them for the classification of the Pictor v.1.1 dataset (Nath, Chaspari, & Behzadan, 2019), while Sumbul and DemİR referenced them in their research. Both models serve as reference points in various studies due to their widespread usage and effectiveness (Sumbul & Demir, 2020).

ResNet, short for residual network, introduced a shortcut connection into convolution layers, acting as an identity mapping to alleviate the training error caused by the gradient vanishing problem (He, Zhang, Ren, & Sun, 2016). Benefiting from their shorter network, ResNet34 and ResNet50 are commonly used variants in current research. For example, Huang et al. conducted an analysis of Chest X-rays ("Deep Transfer Learning for the Multilabel Classification of Chest X-ray Images", 2022), while Pan et al. performed experiments on fundus fluorescein angiography ("Multi-label classification of retinal lesions in diabetic retinopathy for automatic analysis of fundus fluorescein angiography based on deep learning", 2019), both utilizing these models.

In addition to these two classic models, there are currently more pretrained models widely used in multi-label tasks. MobileNet, for instance, utilizes depth-wise separable convolutions, dividing convolution layers into depthwise convolutions and then combining their outputs using a 1*1 pointwise convolution (Howard et al., 2017). This approach reduces model size and enhances inference time. MobileNetV2, an evolution of MobileNets, introduces inverted residuals and linear bottlenecks (Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018). Inverted residuals ensure efficient memory usage, while linear bottlenecks replace non-linear functions in projection convolution, reducing the loss of useful features during dimensionality reduction (Sandler et al., 2018)

Further variations of MobileNet have been developed for various purposes, such as Thin MobileNet by Sinha and El-Sharkawy (Sinha & El-Sharkawy, n.d.), Fd-mobilenet by Qin et al. (Tan et al., 2019), and Mobile-Former by Chen et al (Y. Chen et al., 2022).

MnasNet, another mobile CNN model, reduces size and latency compared to MobileNets by employing a multi-objective neural architecture (Tan et al., 2019). It also utilizes a novel factorized hierarchical search space, splitting layers into unique blocks and searching for the best operation and connection for each block based on the model's input and output shape (Tan et al., 2019). This search space enhances layer diversity and achieves a better trade-off between accuracy and latency.

EfficientNet, derived from MnasNet, optimizes accuracy and FLOPS (floating-point operations per second), which serves as an indirect measure of latency rather than directly optimizing for latency itself (Tan & Le, 2019a). This focus on optimizing FLOPS suggests that EfficientNet exhibits broader applicability across various hardware devices. Initially, EfficientNet-B0

was developed, and subsequently, EfficientNet-B1 to B7 were derived from EfficientNet-B0 through compound model scaling.

Taking into account factors such as higher accuracy, smaller size, and faster inference time, our study selects EfficientNet to process image data. Then, We aim to further evaluate models ranging from EfficientNet-B0 to B7 to determine the best-performing model.

3.2 Methods for Improve Model Performance

Correlation between labels is a long-term research point in the multi-label image classification area to research and leverage solutions to further enhance the model performance on multi-label classification. The attitude of previous research on label correlation can be either to ignore label correlations and train a binary classifier for each label (Sandler et al., 2018), or to let the model learn label dependencies. And the latter method can be further classified into:

1. Ranking labels based on relevance and capturing label dependencies using a loss function, which often relies on a fine-tuned threshold (Sinha & El-Sharkawy, n.d.; Tan et al., 2019; Tan & Le, 2019a).

2. Learning and utilizing label dependencies as constraints to improve classification accuracy (Xu, Li, & Zhou, 2013; Dembczynski, Cheng, & Hullermeier, 2010).

3. Integrating probabilistic classifiers to learn the relationships among labels and determine the probability of correlation between each pair of labels (Nath et al., 2019; Sumbul & Demir, 2020),

4. Integrating CNN model with RNN or a transformer to learn the relationship between low-dimension image features and caption features of labels, thereby, understanding the correspondence between labels, for example, a joint architecture of CNN-RNN was designed by Wang et al ("CNN-RNN: A Unified Framework for Multi-label Image Classification", 2016).

5. Applying a self-attention mechanism to indirectly learn label dependencies by capturing relationships between relevant features from different positions (Howard et al., 2017).

The other improvement discussion mainly focuses on extraction and refinement of potential useful features, like small scale objects and trail labels, and the correlation of labels as well. For instance, the feature refinement network (FRN), designed by Yan et al and applied in Feature Attention Network (FAN) solution, refines and preserves features related to small yet crucial objects and features highly pertinent to infrequent labels by recalibrating the weight of feature, thereby avoiding these important features vanish after passing ReLu activation function (Howard et al., 2017). Similarly, a transformer-based feature learning network is designed to learn salient object features and refine potential useful features such as features of small key objects ("Pattern Recognition", 2022). Meanwhile, the concept of spatial and semantic transformer (SST) is described to extract the spatial correlation of features and semantic correlations between labels, designed by Chen et al (Z.-M. Chen et al., 2022)

With the objective of maintaining a simple and stable model architecture while addressing the trade-off between model size, complexity, and performance, our study proposes to combine EfficientNet with a small transformer model, TinyBert ("ArXiv Paper", 2022). This integration aims to leverage the strengths of both models: EfficientNet for extracting visual features from images and TinyBert for extracting spatial and semantic features from caption data. By doing so, we aim to improve overall model performance.

Furthermore, our model seeks to handle label dependencies by learning combined features extracted by EfficientNet and TinyBert. These combined features are expected to encapsulate the relationships between object features from different positions within the image and the spatial and semantic features extracted from the caption data. This approach allows the model to better understand the potential co-existence of labels, addressing challenges associated with label dependency as the idea discussed in methods 4 and 5.

4 Techniques

4.1 Data Preprocessing

4.1.1 Transformations

In our scenario, we have 18 unique categories ranging from 1 to 19, excluding 12. To handle these multiple labels, the label data is transformed into a one-hot encoded format, as discussed in (Elisseeff & Weston, 2002). This format creates a binary vector where each element represents a possible unique category within the set. Only the element corresponding to the labels present in the sample is set to 1, while the rest are set to 0. The number of "1"s in the vector indicates the number of categories for the sample, and the position of each "1" reflects which categories the sample belongs to. This step serves to clarify the presence of categories for the model to comprehend, similar to methods described in (Wang, Huang, & Ding, 2009). It also enhances the independence of predictions for each category, enabling the model to predict categories simultaneously, an approach reinforced by (Dembczynski et al., 2010).

Furthermore, all images serving as input for training are resized to (256, 256) to match the dimensions of the network, following best practices from ("CNN-RNN: A Unified Framework for Multi-label Image Classification", 2016). Next, 50% of the images are randomly selected and horizontally flipped, and then some are randomly rotated clockwise or anticlockwise by up to 10 degrees, similar to data augmentation techniques discussed in ("Multi-Label Image Classification by Feature Attention Network", 2019). These rotations can lead to the loss of features at the corners of the images. These data augmentation techniques aim to enhance the variability of features within the training dataset while also providing robustness and generalization for the model, and helping to avoid overfitting issues, as explored in ("Pattern Recognition", 2022).

Subsequently, the images are transformed into PyTorch tensor format and normalized, scal-

ing the data to the range [0, 1]. This normalization ensures consistent data scaling, thereby improving convergence speed and preventing exploding gradients during the training process, a key aspect covered in (Z.-M. Chen et al., 2022).

4.1.2 Train-Validation Split

In the study, the best model will be selected based on its accuracy on validation data, which is obtained by partitioning 20% of the training dataset. This portion remains unused during the training phase. The primary objective of employing validation data is to evaluate the model's generalization capability and its effectiveness in handling unseen data, which is assumed to resemble real-world datasets. This validation process ensures that we can choose the model that not only fits the dataset well but also exhibits high robustness, a strategy echoed in (Guo & Xue, 2013), thereby increasing the reliability of the selected model for practical use.

4.2 Principle of Technologies

4.2.1 Loss Function

In this multilabel classification task, Binary Cross-Entropy (BCE) loss combined with the logistic sigmoid function is used as the loss function and activation function respectively. The logistic sigmoid function is used as the activation function of our model. The sigmoid function will convert each raw output of the network to a probability between 0 to 1. It represents the likelihood that the predicted label is a true label. The function of sigmoid function can be defined as(Topper, 2023):

$$\delta(x) = \frac{1}{1+e^{-x}} \tag{1}$$

The Binary Cross Entropy (BCE) is a technique which is used to calculate the loss in deep learning. It measures the difference between the true label and the label predicted by the deep learning model and improves the training process of the model through making penalizations to the wrong predictions. This technique is suitable to be used in multilabel classification tasks, as it can measure the difference between the predictions and the actual labels for each class individually. The BCE loss of an instance can be calculated by(Saxena, 2021):

$$\text{BCE loss function} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1-y_i)\log(1-p_i)] \tag{2}$$

- N is the number of labels
- p_i is the predicted probability for a label
- y_i is the true label for the class

4.2.2 Optimizer

Adam (Adaptive Moment Estimation) is an optimization algorithm used for training deep learning models. It combines the strengths of RMSProp and AdaGrad by adaptively adjusting the

learning rate of each parameter to improve training performance. Adam adjusts the learning rate using past gradient information at each update step(Kingma & Ba, 2014).
In the following part, we will introduce how Adam works:
First, compute the current gradient of the parameters, indicating how to adjust the parameters to reduce the loss. Then, Adam records the exponentially weighted average of past gradients, called the first moment estimate. This smooths the gradients and prevents overly drastic updates.

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t \tag{3}$$

Adam also records the exponentially weighted average of the squared past gradients. It also called the second moment estimate these records.

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\mathbf{g}_t^2 \tag{4}$$

Adam can adjust the learning rate for each parameter, adapting to different gradient variations. Using momentum and adaptive learning rates, Adam updates the parameters, making the optimization process more stable and faster.

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t} \tag{5}$$

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \varepsilon} \tag{6}$$

In summary, Adam is an ideal choice for our multi-label image classification tasks due to its adaptability and efficiency. It simplifies parameter tuning, improves training efficiency and model performance. That's why we choose Adam.

4.2.3 ResNet50

ResNet50 is a deep convolutional neural network (CNN) introduced by Kaiming He and colleagues in 2015. It consists of 50 layers, including convolutions, pooling, and fully connected layers. This architecture was designed to address the vanishing and exploding gradient problems that impede the training during very deep networks.

The core feature of ResNet is the use of residual blocks. In traditional deep neural networks, as the depth of the network increases, the gradients can either vanish or explode as they propagate back through the layers during training. This makes it difficult for the network to learn and converge.

ResNet solves this problem by introducing skip connections, also known as shortcut connections, that bypass one or more layers. Instead of go through each block layer by layer directly, the network learns the residual mapping. This is mathematically represented as:

$$\mathbf{y} = \mathscr{F}(\mathbf{x}, \{W_i\}) + \mathbf{x} \tag{7}$$

Here, $\mathbf{x}$ is the input, $\mathscr{F}(\mathbf{x}, \{W_i\})$ represents the residual mapping to be learned, and $\mathbf{y}$ is the output of the residual block. The skip connection adds the input $\mathbf{x}$ directly to the output of the

residual mapping, which helps to preserve the gradient flow through the network and makes training easier.

The structure of a typical residual block in ResNet includes two or three convolutional layers, each followed by batch normalization and a ReLU activation function. The identity mapping is then added to the output of these layers.

ResNet50 consists of several stages, each containing a different number of residual blocks:

- The first stage includes a convolutional layer followed by a max pooling layer.

- The second stage has 3 residual blocks, each containing 3 convolutional layers.

- The third stage has 4 residual blocks.

- The fourth stage has 6 residual blocks.

- The fifth stage has 3 residual blocks.

- The network ends with a global average pooling layer, followed by a fully connected layer that outputs the final classification scores(He, Zhang, Ren, & Sun, 2015).

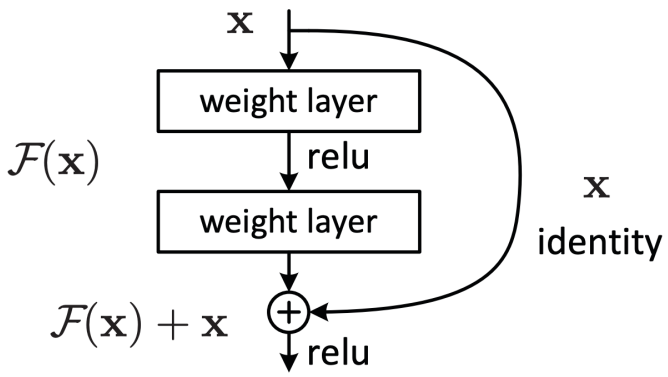

Figure 1: Residual learning: a building block

The introduction of residual blocks in ResNet allows the network to be trained more efficiently when it is very deep. Given these characteristics, ResNet is particularly well-suited for image classification tasks, making it an ideal choice for our classification task. The robust performance and ability to effectively train deep networks are the reasons why ResNet was our first choice for this method.

4.2.4 MobileNetV2

The MobileNetV2 is a kind of convolutional neural network which is lightweight and designed to work efficiently on mobile devices by Google. This model is tried to get a high accuracy while maintaining a low computational cost, which is essential for importing the neural network methodology to mobile devices. The idea of MobileNet is derived from applying "depthwise separable convolutions", which is build up with a depthwise convolution and a pointwise convolution(Bhargava, 2018).

The depthwise separable convolution is firstly introduced in MobileNetV1, which is the previous version of MobileNetV1. It is a technique which greatly reduces the complexity and size of the model so that it makes this model become especially suitable for devices with low computational power (Singh, 2018).

The inverted residual structure is an improved module which was introduced in MobileNetV2, which is one of the most important components in the model. It withdraws the non-linearities in narrow layers and contains a bottleneck structure which increases the number of channels before performing a depthwise separable convolution and enabling the model to learn from more complex features and improve its power in representation (Sharma, 2023).

The bottleneck design of the MobileNetV2 minimizes the requirement for computational power through applying convolutions which can decrease the number of channels before performing depthwise separable convolutions. It makes the model get a great balance between the size of the model and performance (Sharma, 2023).

Compared with Resnet, the MobileNetV2 is designed to be used in a different scenario. It is especially designed to work efficiently on mobile devices. Through the use of inverted residual blocks, linear bottlenecks and depthwise separable convolutions of the model, the requirement for the computational cost is decreased while maintaining a good performance. Also, it applies ReLU6 as the activation function, which allows adjusting with width and resolution multipliers (Bhargava, 2018). By contrast, the Resnet has a deeper architecture with residual blocks to alleviate the gradient vanish problem. Therefore, the MobileNetV2 can be a better choice when we don't have enough computational resources or when we want to build a model with a small size of storage space.

4.2.5 TinyBert

Bert, short for Bidirectional Encoder Representations from Transformers, is a transformer-based model introduced by Google in 2018. It is designed to pre-train deep bidirectional representations by jointly conditioning on both left and right context in all layers. Thus, Bert can understand the context of a word based on its surrounding words. In general, Bert comes in different sizes, with Bert-base being one of the most commonly used variants, consisting of 12 layers, 768 hidden units, and 110 million parameters(Devlin, Chang, Lee, & Toutanova, 2018). However, it is too heavy for our task, we finally choose Tiny Bert as our text encoder.

TinyBert is a transformer-based model that has evolved from Bert-base, enabling it to handle both text transformation and classification tasks. Unlike other researched models that adjust architecture and parameters based on Bert-base and are trained using specialized datasets, TinyBert is directly derived from Bert-base.

A two-stage learning framework, based on the teacher-student paradigm, is applied to develop TinyBert. In this framework, the original Bert, without fine-tuning, serves as the teacher for the first stage, while the fine-tuned Bert acts as the teacher for the second stage. TinyBert is initialized at the beginning of the first stage and acts as a student, aiming to replicate the functions and behaviors of the teachers. A transformer distillation method is designed and leveraged to distill knowledge from both Bert-base and fine-tuned Bert. This method is derived from knowledge distillation (KD), a training process that transfers knowledge from a large-scale and complex model to a smaller, simpler one.

To reduce the network size, the learning framework is designed with a triple-layer mapping function (Equation 1, where 'n' represents the number of layers for the teacher network and 'm' represents the number of layers for the student network). This means that each layer within TinyBert aims to learn and function as three layers from its teacher during the learning process, thereby simplifying the architecture within TinyBert.

$$n = g(m) = 3 \cdot m \tag{8}$$

Compared with Bert-base, TinyBert is 7.5 times lighter with 14.5 M parameters and has 9.4 times faster inference time, while maintaining a performance highly similar to Bert-base. Furthermore, when compared with models of similar size such as BertTiny (14.5M parameters), Bertsmall (29.2M parameters) and MobileBertTiny (15.1M parameters), TinyBert exhibits significantly higher average performance when tested on the General Language Understanding Evaluation (GLUE) official benchmark .

In our study, TinyBert is applied to process and extract features from caption data. This application allows the model to maintain a smaller size and require fewer computational resources to operate, increasing its feasibility. Consequently, it enables a wider application of the model on devices with limited computational power and storage space. Additionally, TinyBert offers faster inference speed on feature extraction compared to any other Bert-base transformer, enhancing the model's efficiency and responsiveness.

4.2.6 EffeicentNet

EfficientNet is a CNN based mobile model that was generated from MnasNet which shows a good balance between accuracy and inference time. Similar to MnasNet, EfficientNet model applied a factorized hierarchical search space to improve more accuracy and reduce latency by diversifying layers in the network. This search space splits original layers into unique blocks, searches the best operation in each block and connects the best operation layers as the model network, thereby the architecture of EfficientNet can be various based on different input and output shapes.

Also, in this search space, there is a trade-off between model accuracy and FLOPS (Equation2). In this equation, ACC(m) and FLOPS(m) represent the accuracy and FLOPS of model m, respectively. T represents the target FLOPS, and the hyperparameter w = -0.07 is utilized to control this trade-off.. The utilization of FLOPS from EfficientNet instead of latency, as seen in its original model MnasNet, has further enhanced the model's generalization across various applications and has made it compatible with a wide range of devices.

$$ACC(m) \cdot [FLOPS(m)/T]^w \tag{9}$$

Compared with other CNN-based models, EfficientNet demonstrates higher accuracy. Additionally, when compared with scaled versions of ResNet50 and MobileNetV1 and V2 using the same scaling method, while they achieve relatively close accuracy, they exhibit longer inference times than EfficientNet. Therefore, our study applies EfficientNet to process image features after considering the trade-off between model performance and size.

Moreover, considering EfficientNet can be further scaled from EfficientNet-B0 to B7, where EfficientNet-B1 to B7 are adaptations of EfficientNet-B0 through a compound model scaling process, these models exhibit similar performance on transfer learning datasets and will be further tested to select the best-performing one. Specifically, considering the larger parameter numbers of 30M, 43M, and 66M respectively, and longer inference times of 9.9B, 19B, and 37B for models EfficientNet-B5, B6, and B7, our choice can be further narrowed down to EfficientNet-B0 to B4. Among these five models, the one that performs the best within our model network and results in the highest accuracy will be finally chosen as our ultimate selection.

4.2.7 Justification of the Model Structure

Our model, named EfficientTinyBERTMultiModal, integrates the power of both EfficientNet and TinyBERT architectures. EfficientNet is employed to efficiently extract and process image features, while TinyBERT is utilized to extract and process text features.

In our architecture, image data undergoes processing through the EfficientNet model, which specializes in image feature extraction. Simultaneously, textual data is fed into the TinyBERT model, known for its prowess in understanding textual and semantic context. The output of both models is then combined by concatenating the extracted features. To fuse the image and text features effectively, we employ a ReLU activation function followed by a dropout layer to prevent overfitting. Finally, the combined features are passed through a linear classifier, allowing the model to learn the intricate relationships between the image and text representations and make predictions across multiple labels.

5 Experiment and Analysis

5.1 Environment Configuration

This project was conducted entirely on google Colab, using Python 3 and operating on Google Compute Engine. Due to the high resource demands of this project, it is necessary to subscribe to Colab Pro or purchase additional compute units to use GPUs of L4 or higher. If opting to run locally, a GPU is also essential. The configuration for this experiment included 80GB of RAM, a 40GB GPU, and 120GB of disk space.

5.2 Evaluation Metrics

The evaluates metrics for this study is **F1 score**, and **time efficiency**.

$$Precision = \frac{TP}{Tp + FP} \tag{10}$$

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{12}$$

According to equation 3 and 4, Precision measures the accuracy of the positive predictions (i.e., how many of the predicted positive labels are actually correct), while recall measures the model's ability to identify all relevant instances (i.e., how many of the actual positive labels are captured by the model). The F1 score is the harmonic mean of precision and recall. It ensures that both precision and recall are equal important, preventing the model from being too biased towards either high precision or high recall. In our imbalanced dataset, Label 1 have significantly more instances than others. This imbalance can skew metrics like accuracy that makes the model seem more effective than it actually is. Thus, we choose F1 as our final parameters.

5.3 Comparison Experiment

5.3.1 Compare of performance of different Model

In this section, we present a comparative analysis of various machine learning models to identify the optimal configuration for our application requirements. We evaluate models based on their F1-score, training and prediction times, and model size, as detailed in the accompanying table. Given constraints on model size, we seek efficient alternatives to the top-performing models, which exceed our size limitations. Our analysis includes a detailed evaluation of various EfficientNet and BERT model variants. We conclude with our selection of EfficientNet b0 and TinyBERT, which provide a balanced trade-off between performance and size, adhering to our application's constraints.

Table 1: Performance of different model

	Validation F1-score	Training Time	Predict Time	Size
Resnet50	0.7894	12 min26s	48.48s	90.15 MB
MobileNetV2	0.7718	7 min56s	35.80s	8.84 MB
EfficientNetb0	0.7960	12min14s	44.86s	15.70 MB
EfficientNetb1	0.8042	15min11s	51.46s	25.40 MB
EfficientNetb2	0.8014	15min52s	53.33s	29.96 MB
EfficientNetb3	0.8146	19min44s	61.74s	41.50 MB
EfficientNetb4	0.8163	25min4s	73.29s	67.86 MB
Bert Base	0.8204	13min41s	32s	417.82 MB
Efficientb0TinyBERT	0.8348	25min16s	2min21s	73.57 MB
Efficientb1TinyBERT	0.8419	27min11s	2min27s	83.28 MB
Efficientb2TinyBERT	0.8483	27min8s	2min26s	88.08 MB
Efficientb3TinyBERT	0.8528	29min13s	2min28s	99.86 MB

According to the comparison experiment, we can find the Bert and EfficientNet 04 has the highest F1 score in the final prediction. However, due to the size limitation of 100M, we cannot choose the EfficientNet 04 and Bert which are more than 250M. Thus, our approach for this problem is to find the smaller alternative for each model.

For EfficientNet, we do the research and find the size of EfficientNet b4 is 19M while the size of EfficientNet b0 is only 5.3M(Tan & Le, 2019b). Hence, we choose the EfficientNet b0 as our final image encoder.

For text encoder, the regular bert is 108M, but the small and tiny bert can be as small as 30M(Bigelow, 2022). Then we try different version of bert, we finally find a tiny bert called TinyBERT_General_4L_312D on huggging face that is 7.5x smaller and 9.4x faster on inference than BERT-base and achieves competitive performances in the tasks of natural language understanding(Jiao et al., 2019). Therefore, we choose this tiny bert as our final text encoder.

5.4 Hyperparameter Optimization

In this section, it provides a detailed analysis of the hyperparameter chosen for our base model. By adjusting these parameters, we aim to find the most effective settings that maximize the model's predictive accuracy and minimize computational cost.

5.4.1 Base Model

Hyperparameter Analysis of Base Model

Table 2: Hyperparameter Of Base Model

Epoch	3
Batch_size	16
Learning Rate	0.0001

The base model is designed as a multi-modal system, incorporating EfficientNet B3 for image encoding and TinyBERT for text encoding. This combination was selected based on its superior performance in preliminary comparative experiments. Before delving into hyperparameter optimization, it's essential to understand the initial settings derived from typical configurations used in similar models. These settings are pivotal as they provide a stable starting point for further refinement aimed at enhancing the model's performance.

Epochs set to 3. Since the model is large and the dataset may also be substantial, too many training epochs not only consume a lot of time but could also lead to overfitting. Limiting the number of epochs helps ensure sufficient learning while controlling training time and preventing overfitting.

Batch Size Initially set at 16. This is a relatively small batch size, which helps the model learn more stably in the early stages of training, especially in terms of resource limitations and model response speed. A smaller batch size reduces memory consumption and allows the model to update its weights more frequently during training, facilitating a smoother optimization process.

Learning rate set at 0.0001. A lower learning rate ensures that the model does not overshoot the minimum point as it approaches the optimal solution, thus ensuring smooth convergence in the later stages of training. While the learning speed might be slower, this helps achieve more refined learning outcomes, especially with complex models and datasets.

Result Analysis

Table 3: Result For Base Model

F1	Training Time(min)	Predict Time(min)
0.851	35.33	3.57

According to Figure 2, the F1 score and loss of our model stabilize significantly after the initial epoch. As presented in Table 3, the base model achieves an F1 score of 0.851 on the validation set when using default parameters, demonstrating robust performance. Furthermore, the training process is completed in 35.33 minutes, and the model requires only 3.57 minutes to

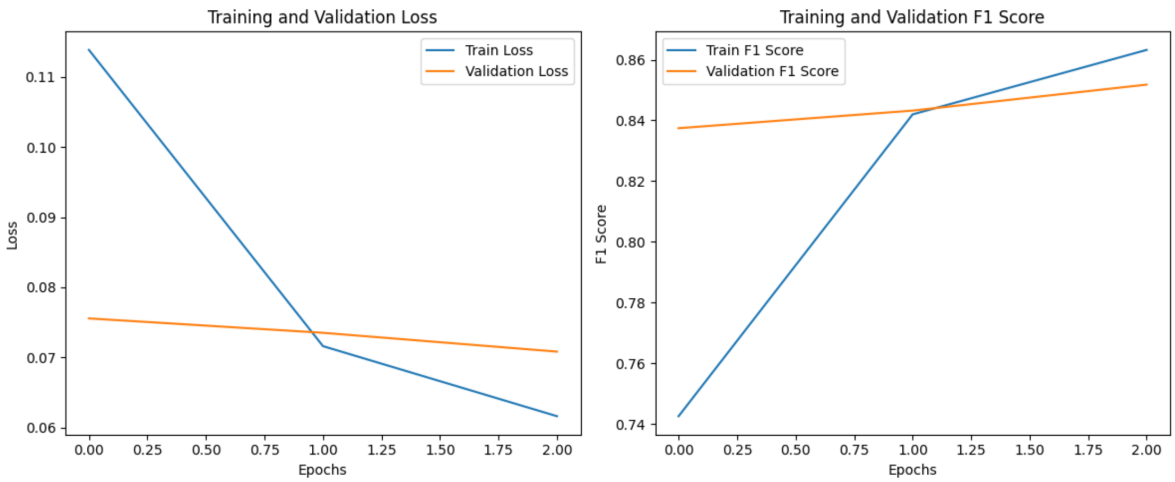

Figure 2: Loss and F1 of Base Model

make predictions, highlighting its efficiency. These findings indicate that our model's architectural design and parameter settings are effectively tailored to address the research challenges, offering practical advantages and substantial potential for real-world applications.

5.4.2 Batch Size optimizing

In this part, we will evaluate the performance by using different batch sizes. Batch size has high influence for both the training time and F1 score. It refers to the number of examples that the model works through before it updates its weights during training.

Table 4: Set Up for Batch Size

Batch size	16	32	64

Table 5: Other Hyperparameter in Batch Size Optimizing experiment

Epoch	3
Learning rate	0.0001

According to Table 6, the performance of our model varies significantly with changes in batch size. With a batch size of 16, the model achieves the highest F1 score of 0.854, indicating optimal learning and generalization at this scale. However, this setting also results in a longer training time of 16.73 minutes, suggesting a trade-off between performance and efficiency. Increasing the batch size to 32 results in a slight decrease in F1 score to 0.847 but reduces the training time significantly to 23.95 minutes. Further increasing the batch size to 64 slightly improves the F1 score to 0.851 but drastically cuts down the training time to just 0.924 minutes,

which is advantageous for scenarios where rapid model training is prioritized over minor improvements in performance. Thus, selecting an appropriate batch size depends on the specific requirements for training efficiency versus performance accuracy.

Table 6: Performance on Different Batch Size

Batch Size	F1	Training Time(min)	Test Time(min)
16	0.854	16.73	3.57
32	0.847	23.95	2.24
64	0.851	35.33	1.17

5.4.3 Learning Rate optimizing

In this part, we will evaluate the performance by using different learning rate. A proper learning rate will fasten the speed of convergence. It refers to step size for updating parameters to minimize loss at each iteration.

Table 7: Options for Learning Rate

Learning Rate	0.00001	0.0001	0.001

Table 8: Other Hyperparameter for Learning Rate optimizing

Epoch	3
Batch_size	16

Table 9: Performance on Different Learning Rate

Learning Rate	F1	Training Time(min)	Test Time(min)
0.00001	0.630	15.9	1.2
0.0001	0.854	16.73	1.14
0.001	0.784	15.92	1.14

From Table 9, it is evident that the learning rate significantly influences the model's performance and training dynamics. A learning rate of 0.0001 yields the best results with an F1 score of 0.854, highlighting its efficiency in guiding the model towards optimal convergence without overshooting. Additionally, the training time at this learning rate is 16.73 minutes, which is

marginally higher compared to other rates but justifiable given the improved accuracy. At a lower learning rate of 0.00001, the model's F1 score drops significantly to 0.630, indicating that the slower adjustments in the model parameters are too conservative, possibly causing the model to get stuck in local minima or taking too long to converge. Conversely, increasing the learning rate to 0.001 improves convergence speed slightly as indicated by the consistent test time of 1.14 minutes but at the cost of reduced accuracy, with the F1 score decreasing to 0.784. This suggests that while the model converges faster, it potentially skips over optimal solutions.

5.4.4 Epoch optimizing

In this part, we will evaluate the performance by using different epochs. A good selection of epoch will get a relative stable accuracy without wasting resource. It refers to number of times the algorithm works through the entire dataset. An epoch includes every iteration over all training data.

Table 10: Options for Epoch

Epoch	2	3	4

Table 11: Other Hyperparameter for Epoch optimizing

Batch_size	16
Learning Rate	0.0001

Table 12: Performance on Different Epoch

Epoch	F1	Training Time(min)	Test Time(min)
4	0.852	21.68	1.25
3	0.854	16.73	1.14
2	0.852	15.92	1.14

From Table 12, it is clear that the number of training epochs plays a critical role in balancing model accuracy and resource utilization. The optimal performance is achieved with 3 epochs, where the model attains an F1 score of 0.854. This setting also results in a training time of 16.73 minutes, which is efficient compared to the longer duration required for 4 epochs. Increasing the epochs to 4 results in a slightly decreased F1 score of 0.852 and a higher training time of 21.68 minutes, suggesting diminishing returns with additional training beyond the third epoch. Conversely, reducing the epochs to 2 maintains a similar F1 score of 0.852 but further decreases the training time to 15.92 minutes. However, the minimal improvement in training time does not justify the potential risk of underfitting, as evidenced by the stable F1 score when using 3 epochs.

5.5 Ablation Experiment

In this section, we describe the ablation experiments conducted to evaluate the contributions of different components in our model. Our original model is a multimodal model combining EfficientNet B0 and Tiny BERT. Therefore, the ablation experiments involve testing the performance of EfficientNet B0 only and Tiny BERT only, while keeping all other parameters constant.

5.5.1 EfficientNet b3

We use only the image encoder, which is EfficientNet B0, to perform our image multi-label classification. We compare the results to determine whether using only the image encoder is better or if our combined image and text model have better performance.

Table 13: Hyperparameter Of Model

Epoch	3
Batch_size	16
Learning Rate	0.0001

Table 14: Performance on Unimodal and Multimodal Models

models	F1	Training Time(min)	Test Time(min)
EfficientNet b3 + Tiny BERT	0.854	16.73	1.14
EfficientNet b3	0.815	17.91	1.01

From the performance comparison in the table, we observe that the multimodal model, integrating EfficientNet B3 and Tiny BERT, achieves a higher F1 score of 0.854, compared to the unimodal model with only EfficientNet B3, which scores an F1 of 0.815. This demonstrates the added value of incorporating textual analysis alongside image processing, as it enhances the model's ability to interpret and classify multi-faceted data more accurately. Moreover, despite the multimodal model requiring a slightly longer training time of 16.73 minutes, compared to 17.91 minutes for the unimodal model, the improved performance justifies the minimal increase in computational time. The test time also shows minimal difference, with the multimodal model at 1.14 minutes and the unimodal model at 1.01 minutes.

5.5.2 Tiny Bert

In this experiment, we aim to explore the performance of using only the text encoder, which is Tiny BERT, for our multi-label classification task. We compare the results to determine whether

using only the text encoder is better or if our combined image and text model has better performance.

Table 15: Hyperparameter Of Model

Epoch	3
Batch_size	16
Learning Rate	0.0001

Table 16: Performance on Unimodal and Multimodal Models

models	F1	Training Time(min)	Test Time(min)
EfficientNet b3 + Tiny BERT	0.854	16.73	1.14
Tiny BERT	0.805	6.64	0.3

According to the performance data, the multimodal model (EfficientNet B3 + Tiny BERT) outperforms the Tiny BERT-only model. The multimodal model achieves an F1 score of 0.854, with a training time of 16.73 minutes and a test time of 1.14 minutes. On the other hand, the Tiny BERT model achieves an F1 score of 0.805, with a shorter training time of 6.64 minutes but a significantly longer test time of 29 minutes.

These results indicate that while the unimodal Tiny BERT model has a shorter training time, it does not reach the same performance level as the multimodal model in terms of F1 score. The multimodal model effectively leverages both image and text data, leading to superior performance in this multi-label classification task. This underscores the effectiveness of integrating both image and text models for enhanced accuracy and reliability in predictions.

5.6 Best Model

The best model, trained with a combination of 3 epochs, a batch size of 16, and a learning rate of 0.0001, achieves an impressive F1 score of 0.8966. This high F1 score indicates that the model effectively balances precision and recall, resulting in robust performance for multi-label classification tasks. The training time for this model is 29.37 minutes, which is a reasonable duration given the complexity and the resulting high performance. Additionally, the test time is 2.17 minutes, showing that the model is efficient during the inference phase.

Figure 3 further illustrates the stability and convergence of the model, with loss decreasing consistently and accuracy improving across the training epochs. This indicates that the chosen hyperparameters are well-suited for optimizing the model's performance without overfitting.

Table 17: Hyperparameter Of Model

Epoch	3
Batch_size	16
Learning Rate	0.0001

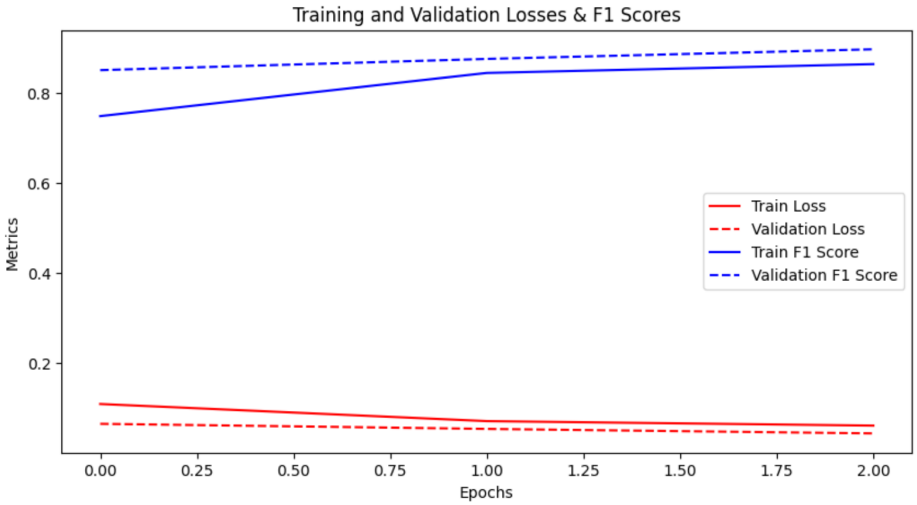

Figure 3: Loss and Accuracy of Best Model

Table 18: Performance on Best Model

F1	Training Time(min)	Test Time(min)
0.8966	29.37	2.17

Overall, the combination of EfficientNet B3 and Tiny BERT, fine-tuned with these hyperparameters, proves to be the most effective configuration. The integration of both image and text data, alongside optimized training settings, ensures that the model can accurately and efficiently handle multi-label classification tasks.

6 Summary and Reflection

In this multi-label classification task, we firstly download the train and test set from Kaggle, make some Exploratory Data Analysis (EDA), and split the train set into training and validation set. After that, a comprehensive evaluation has been conducted to image-based models including ResNet50, MonileNet V2, EfficientNet B0 to B4, text-based models including BERT and Tiny BERT and multimodal models which integrate EfficientNet B0 to B3 with Tiny BERT. Through the experimentation and comparison, the combination of EfficientNet and Tiny BERT showed the highest F1-score with batch size of 16, learning rate of 0.00001 and epoch number of 2. This finding shows that the integration of visual data and textual data can actively increase

the performance of models in multilabel classification tasks.

In conclusion, this study underscores the importance of applying a suitable model architecture to the specific data types, and it proves that through integrating the CNN-based model for visual data and transformer-based models for textual data, coupled with hyperparameter tuning, deep learning model can have a great performance in multilabel classification tasks. The understanding and insights bring us a solid base to deep learning research and the future investigation of Artificial Intelligence.

References

Arxiv paper. (2022). Retrieved from https://doi.org/10.48550/arXiv.1909.10351

Bhargava, C. (2018). *Mobilenetv2: Inverted residuals and linear bottlenecks.* Retrieved from https://towardsdatascience.com/mobilenetv2-inverted-residuals-and-linear-bottlenecks-8a4362f4ffd5 (Accessed: 2024-05-17)

Bigelow, R. (2022). Bert. Retrieved from https://ecampusontario.pressbooks.pub/conversationalai/chapter/bert/

Chen, Y., Dai, X., Chen, D., Liu, M., Dong, X., Yuan, L., & Liu, Z. (2022). Mobile-former: Bridging mobilenet and transformer.. Retrieved from https://openaccess.thecvf.com/content/CVPR2022/html/Chen_Mobile-Former_Bridging_MobileNet_and_Transformer_CVPR_2022_paper.html

Chen, Z.-M., Cui, Q., Zhao, B., Song, R., Zhang, X., & Yoshie, O. (2022). Sst: Spatial and semantic transformers for multi-label image recognition. Retrieved from https://doi.org/10.1109/TIP.2022.3148867

Cnn-rnn: A unified framework for multi-label image classification. (2016). Retrieved from https://openaccess.thecvf.com/content_cvpr_2016/html/Wang_CNN-RNN_A_Unified_CVPR_2016_paper.html

Cnn: Single-label to multi-label. (2014). Retrieved from https://doi.org/10.48550/arXiv.1406.5726

Conference on computer vision and pattern recognition. (2009). Retrieved from https://doi.org/10.1109/CVPR.2009.5206816

Deep transfer learning for the multilabel classification of chest x-ray images. (2022). Retrieved from https://doi.org/10.3390/diagnostics12061457

Dembczynski, K., Cheng, W., & Hullermeier, E. (2010). Bayes optimal multilabel classification via probability classifier chains.. Retrieved from https://proceedings.mlr.press/v80/dembczynski10a.html

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. Retrieved from https://arxiv.org/abs/1810.04805

Elisseeff, A., & Weston, J. (2002). A kernel method for multi-labelled classification.. Retrieved from https://papers.nips.cc/paper/2001/hash/4bbaa21ccdf70252c1b5b4987ac9179b-Abstract.html

Guo, Y., & Xue, W. (2013). Probabilistic multilabel classification with sparse feature learning.. Retrieved from https://www.ijcai.org/Proceedings/13/Papers/410.pdf

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. Retrieved from https://arxiv.org/abs/1512.03385

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Retrieved from https://doi.org/10.1109/CVPR.2016.90

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. Retrieved from https://arxiv.org/abs/1704.04861

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., ... Liu, Q. (2019). Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. Retrieved from https://arxiv.org/abs/1412.6980

Multi-label classification of retinal lesions in diabetic retinopathy for automatic analysis of fundus fluorescein angiography based on deep learning. (2019). Retrieved from https://doi.org/10.1007/s00417-019-04575-w

Multi-label image classification by feature attention network. (2019). Retrieved from https://doi.org/10.1109/ACCESS.2019.2929512

Nath, N. D., Chaspari, T., & Behzadan, A. H. (2019). Single-and multi-label classification of construction objects using deep transfer learning methods. Retrieved from https://www.itcon.org/paper/2019/27

Pal, S. (2016). *Transfer learning and fine tuning for cross domain image classification with keras.* Retrieved from https://github.com/keras-team/keras-io/blob/master/examples/vision/ipynb/transfer_learning.ipynb

Pattern recognition. (2022). Retrieved from https://doi.org/10.1016/j.patcog.2022.109203

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. Retrieved from https://openaccess.thecvf.com/content_cvpr_2018/html/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.html

Saxena, S. (2021). *Binary cross entropy/log loss for binary classification.* Retrieved from https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/ (Accessed: 2024-05-17)

Sharma, N. (2023). *What is mobilenetv2? features, architecture, application and more.* Retrieved from https://www.analyticsvidhya.com/blog/2023/12/what-is-mobilenetv2/ (Accessed: 2024-05-17)

Singh, D. (2018). *Review: Mobilenetv2 – light weight model image classification.* Retrieved from https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c (Accessed: 2024-05-17)

Sinha, D., & El-Sharkawy, M. (n.d.). Thin mobilenet: An enhanced mobilenet architecture..

Sumbul, G., & Demir, B. (2020). A deep multi-attention driven approach for multi-label remote sensing image classification. Retrieved from https://doi.org/10.1109/ACCESS.2020.2995805

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile.. Retrieved from https://openaccess.thecvf.com/content_CVPR_2019/html/Tan_MnasNet_Platform-Aware_Neural_Architecture_Search_for_Mobile_CVPR_2019_paper.html

Tan, M., & Le, Q. V. (2019a). Efficientnet: Rethinking model scaling for convolutional neural networks.. Retrieved from https://proceedings.mlr.press/v97/tan19a.html

Tan, M., & Le, Q. V. (2019b). Efficientnet: Rethinking model scaling for convolutional neural networks. Retrieved from https://arxiv.org/abs/1905.11946

Topper, N. (2023). *Sigmoid activation function: An introduction.* Retrieved from https://builtin.com/machine-learning/sigmoid-activation-function (Accessed: 2024-05-17)

Very deep convolutional networks for large-scale image recognition. (2014). Retrieved from https://arxiv.org/abs/1409.1556

Wang, H., Huang, H., & Ding, C. (2009). Image annotation using multi-label correlated green's function.. Retrieved from https://openaccess.thecvf.com/content_cvpr_2009/

html/Wang_Image_Annotation_Using_2009_CVPR_paper.html
Xu, M., Li, Y., & Zhou, Z. (2013). Multi-label learning with pro loss.. Retrieved from https://ojs.aaai.org/index.php/AAAI/article/view/8651