COMP5329 Deep Learning Assignment2

Group members:
Binghang Li(530191027)
Simiao Yu(520428764)
Qiuhan Li(530748315)

Abstract

This study aims to perform multi-label classification using a combination of ResNet50 for image processing and BERT for text processing. The model achieved an impressive F1 score of 94.364 on the Kaggle test set. Various techniques, including data normalization and augmentation, were employed. Comparative analysis showed that BERT significantly outperforms LSTM for text processing, and ResNet50 is more effective than ResNet18 for image feature extraction. Extensive hyperparameter tuning further improved performance. These findings highlight the effectiveness of integrating state-of-the-art models for complex classification tasks.

1 Introduction

1.1 Study Aim

The aim of this study is to perform multi-label classification on a given dataset by implementing various convolutional neural networks (CNNs), including state-of-the-art architectures recognized within the deep learning community. The study focuses on comparing the performance of different DeepConvNets and analyzing the components that enable these models to achieve high performance on this specific problem. Additionally, we explore potential modifications to the networks and data handling techniques that may further enhance their performance. This comprehensive analysis helps identify the best CNN architecture for the dataset and understand how each component contributes to the overall effectiveness of the model.

1.2 Significance of the Study

Multi-label classification is increasingly important in many real-world applications. From tagging photos on social media to diagnosing multiple conditions in medical images, the ability to predict several labels for a single image is crucial. This task helps us understand the complexities of handling and analyzing large datasets, balancing class distributions, and optimizing models to achieve the best results. It also prepares us for real-world scenarios where deep learning engineers need to deploy efficient, pre-trained networks for custom applications and datasets.

1.3 Method Overview and Motivation

In this assignment, we combine the strengths of Convolutional Neural Networks (CNNs) and Transformer models to tackle the multi-label classification task. We use a pre-trained ResNet50 to extract features from images and a pre-trained BERT model to process the accompanying text captions . The features from both the images and the text are then combined to predict the labels.

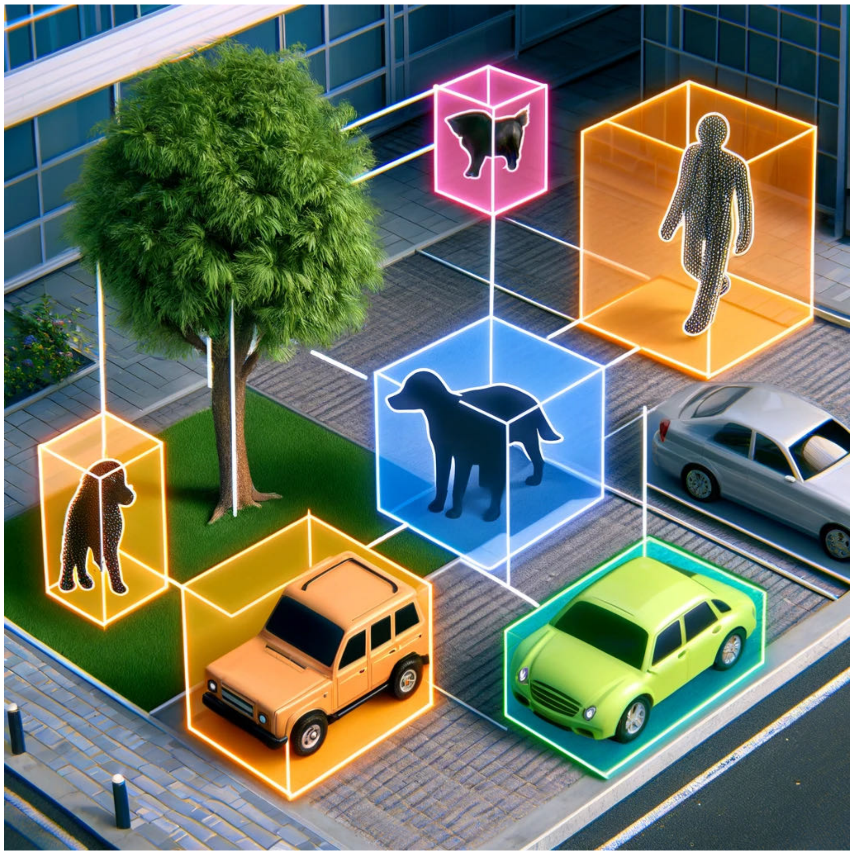


Figure 1: Imgae Classification

We chose this approach because CNNs excel at extracting spatial features from images , while Transformers are highly effective in understanding contextual information from text . By integrating these two models, we can leverage the rich information from both visual and textual data, aiming to improve prediction accuracy. This fusion model approach not only demonstrates the power of combining different deep learning techniques but also provides a practical framework for solving complex multi-label classification problems.

2 Related Work

In the field of multi-label classification, several existing methods and architectures have been proposed and widely studied. This section highlights some of the key related works in this area, focusing on both image-based and text-based approaches, as well as their integration.

2.1 Image-based Approaches

2.1.1 Single Shot MultiBox Detector (SSD)

The SSD framework is widely recognized for object detection tasks, including multi-label classification. SSD discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. The approach combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes. This method is known for its efficiency and accuracy in handling multiple objects and labels within an image [1].

2.1.2 Deep Residual Networks (ResNet)

ResNet, particularly ResNet50, is a powerful convolutional neural network architecture that has been highly effective in image classification tasks. It introduced the concept of residual learning, enabling the training of very deep networks by addressing the vanishing gradient problem. ResNet's architecture has been foundational in many computer vision tasks, including multi-label classification, due to its ability to extract rich hierarchical features from images [2].

2.2 Text-based Approaches

2.2.1 Bidirectional Encoder Representations from Transformers (BERT)

BERT has set new benchmarks in various natural language processing (NLP) tasks due to its deep bidirectional nature, which allows it to capture context from both directions. BERT's ability to understand the context in which words are used makes it highly effective for text classification tasks. For multi-label classification, BERT can be fine-tuned to predict multiple labels based on the context provided by the captions [3].

2.2.2 Label-Dependent Representation

This approach enhances multi-label text classification by incorporating label dependencies directly into the representation of the text. By creating a label-dependent representation, the model can better capture the relationships between different labels, improving classification performance. This method has shown improvements in various benchmarks, especially in datasets with high label cardinality and diversity [4][5].

2.3 Integrated Approaches

2.3.1 CheXNet

CheXNet is an example of a model that successfully integrates image and text data for multi-label classification in the medical domain. It uses a DenseNet architecture pre-trained on ImageNet for feature extraction from chest X-rays, followed by a fully connected layer for predicting multiple conditions. This model demonstrated the potential of combining visual and textual data to enhance the prediction accuracy in complex multi-label scenarios [6].

2.3.2 BERT-CNN Fusion Models

Combining BERT and CNN models leverages the strengths of both architectures. BERT extracts contextual features from text, while CNN captures spatial features from images. This fusion approach has been applied to various tasks, showing improved performance over using either model alone. Such integrated models are effective in multi-label classification tasks that involve both image and text data [7].

3 Techniques

3.1 Normalization

3.2 Min-Max Normalization

This technique scales the pixel values of images from their original range of 0-255 to a new range of 0-1. This is done using the `ToTensor()` function, which converts image data into tensors and normalizes the pixel values. The formula for Min-Max normalization is:

$$x' = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

where x is the original pixel value, x_{min} is the minimum pixel value (0), and x_{max} is the maximum pixel value (255).

3.2.1 Standardization

This method transforms the data to have a mean of 0 and a standard deviation of 1, calculated per channel. This helps in centering the data and reducing the effects of outliers. The formula for standardization is:

$$x' = \frac{x - \mu}{\sigma}$$

where x is the original pixel value, μ is the mean of the pixel values, and σ is the standard deviation of the pixel values.

3.3 Data Augmentation Techniques

3.3.1 Random Horizontal Flip

This technique randomly flips the image horizontally with a probability of 50%. It helps in increasing the variability of the dataset, making the model more robust to different orientations. This is implemented as:

$$\text{image}_{\text{flipped}} = \begin{cases} \text{image}_{\text{original}} & \text{if } p < 0.5 \\ \text{image}_{\text{original}}.\text{flip}(\text{horizontal}) & \text{otherwise} \end{cases}$$

where p is a random number between 0 and 1.

3.3.2 Random Vertical Flip

Similar to horizontal flipping, this flips the image vertically with a 50% probability, adding further variability and robustness.

3.3.3 Convert to RGB

This step ensures that all images are in RGB mode, which is necessary for consistency since some images might be in grayscale. RGB images have three color channels (Red, Green, Blue), each with values ranging from 0 to 255.

3.3.4 Functional Transformations

These transformations include adjustments to brightness, contrast, hue, saturation, and sharpness. Each transformation modifies the pixel values to enhance certain features of the image. For example:

Brightness: Adjusting the brightness involves multiplying the pixel values by a factor.

$$\text{image}_{\text{bright}} = \text{image} \times \text{factor}$$

Contrast: Adjusting contrast changes the difference between the light and dark areas of the image.

$$\text{image}_{\text{contrast}} = (\text{image} - \mu) \times \text{factor} + \mu$$

Hue: Adjusting the hue changes the color balance of the image.

Saturation: Adjusting saturation changes the intensity of colors.

Sharpness: Adjusting sharpness changes the clarity of edges in the image.

These preprocessing steps ensure that the data fed into the model is consistent, normalized, and varied, which helps in improving the model's performance and robustness.

3.4 Tokenization for Text Data

We used the BERT tokenizer to process captions. Tokenization converts text into tokens that BERT can understand. This process includes adding special tokens (`[CLS]`, `[SEP]`), padding to a maximum length, and truncating if necessary. The tokenizer also converts words into word pieces if the word is not found in the vocabulary.

Weighted Approaches

- **WeightedRandomSampler:** Applied inverse label occurrence to balance the dataset. This means the probability of selecting a sample is inversely proportional to the frequency of its label in the dataset.
$$p_i = \frac{1}{f_i}$$
where p_i is the sampling probability for the i-th label, and f_i is the frequency of the i-th label.
- **BCEWithLogitsLoss with pos_weight:** Used binary cross-entropy loss with positive weights to handle imbalance effectively. This adjusts the weight of the positive examples in the loss function.
$$\text{Loss}(x, y) = -w_c \left[y_c \log(\sigma(x_c)) + (1 - y_c) \log(1 - \sigma(x_c))\right]$$
where x_c is the prediction, y_c is the true label, σ is the sigmoid function, and w_c is the weight for class c.

3.5 Loss Functions

We used two key loss functions:

BCEWithLogitsLoss

Combines binary cross-entropy loss and a sigmoid layer for numerical stability.

$$\text{BCEWithLogitsLoss}(x, y) = -\sum_{c=1}^{C} \left[p_c \cdot y_c \log(\sigma(x_c)) + (1 - y_c) \log(1 - \sigma(x_c))\right]$$

where c is the class label, x_c is the model output, y_c is the ground truth, p_c is the class weight, and σ is the sigmoid function.

MultiLabelSoftMarginLoss

Optimizes a multi-label one-vs-all loss based on max-entropy.

$$\text{MultiLabelSoftMarginLoss}(x, y) = -\frac{1}{C} \sum_{i=1}^{C} \left[y[i] \log\left(\frac{1}{1 + e^{-x[i]}}\right) + (1 - y[i]) \log\left(\frac{e^{-x[i]}}{1 + e^{-x[i]}}\right)\right]$$

where C is the number of classes, x are the logits, and y are the labels.

3.6 Model Architecture

We combined ResNet50 and BERT models to leverage both image and text features for multi-label classification:

3.6.1 Image Model (ResNet50)

Extracts image features using convolutional layers.

Residual Block:

$$y = F(x, \{W_i\}) + x$$

where x is the input, y is the output, and F represents the residual mapping using weights $\{W_i\}$.

3.6.2 Text Model (BERT)

Processes captions to extract contextual information.

Attention Mechanism:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

where Q (query), K (key), and V (value) matrices capture relationships between words, and d_k is the dimensionality of the keys.

3.6.3 Fusion Layer

Combines features from both models for final classification.

Feature Combination:

$$\text{combined\_features} = \text{text\_features} + \text{image\_features}$$

where `text_features` are extracted by BERT and `image_features` by ResNet50.

3.7 Model Training and Evaluation

3.7.1 Training Loop

The model is trained over several epochs using the training loop, which feeds batches of data into the model, computes the loss, and updates the model parameters using backpropagation and an optimizer (e.g., Adam).

$$\text{loss} = \text{BCEWithLogitsLoss}(\text{predictions}, \text{labels})$$

3.7.2 F1 Score

Used to evaluate the model's performance, balancing precision and recall.

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

The F1 score is particularly useful in multi-label classification tasks where we need to balance the importance of correctly predicting multiple labels.

4 Experiments and Results

4.1 Model Training and Performance

We trained our model using a combination of ResNet50 for image processing and BERT for text processing. Below are the training results over four epochs:

Model training completed successfully with noticeable improvements in the loss metrics.

4.2 Model Evaluation

We used the trained model to predict the labels of the test images, generating the `predict_output.csv` file. Upon submitting this file to Kaggle, we achieved an impressive F1 score of 94.364.

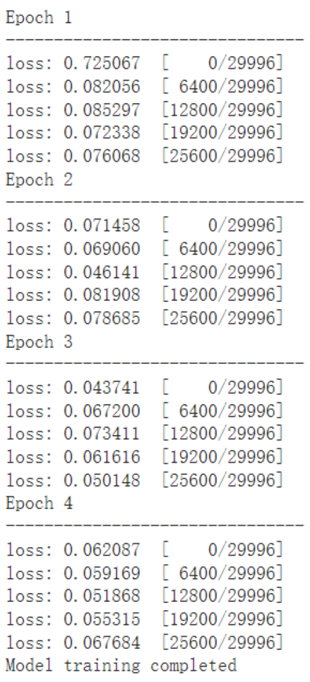
4.3 Comparative Analysis

To ensure the robustness of our approach, we tested several alternative methods:

Configuration	F1 Score
ResNet50 + BERT (Full)	94.364
ResNet50 + LSTM	83.514
ResNet18 + BERT	92.032

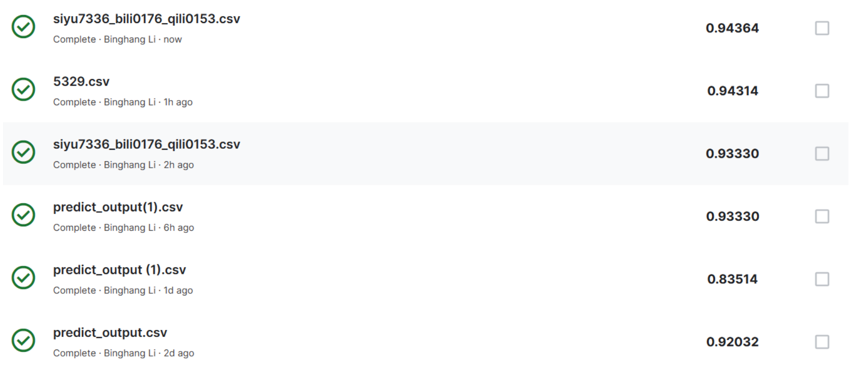
Table 1: Ablation study results.

The results demonstrate that BERT significantly outperforms LSTM in processing textual data, contributing to the higher accuracy of our model.



```
Epoch 1
-------------------------------
loss: 0.725067  [    0/29996]
loss: 0.082056  [ 6400/29996]
loss: 0.085297  [12800/29996]
loss: 0.072338  [19200/29996]
loss: 0.076068  [25600/29996]
Epoch 2
-------------------------------
loss: 0.071458  [    0/29996]
loss: 0.069060  [ 6400/29996]
loss: 0.046141  [12800/29996]
loss: 0.081908  [19200/29996]
loss: 0.078685  [25600/29996]
Epoch 3
-------------------------------
loss: 0.043741  [    0/29996]
loss: 0.067200  [ 6400/29996]
loss: 0.073411  [12800/29996]
loss: 0.061616  [19200/29996]
loss: 0.050148  [25600/29996]
Epoch 4
-------------------------------
loss: 0.062087  [    0/29996]
loss: 0.059169  [ 6400/29996]
loss: 0.051868  [12800/29996]
loss: 0.055315  [19200/29996]
loss: 0.067684  [25600/29996]
Model training completed
```

Figure 2: Training loss



siyu7336_bili0176_qili0153.csv Complete · Binghang Li · now	0.94364
5329.csv Complete · Binghang Li · 1h ago	0.94314
siyu7336_bili0176_qili0153.csv Complete · Binghang Li · 2h ago	0.93330
predict_output(1).csv Complete · Binghang Li · 6h ago	0.93330
predict_output (1).csv Complete · Binghang Li · 1d ago	0.83514
predict_output.csv Complete · Binghang Li · 2d ago	0.92032

Figure 3: F1 Scores

4.4 Hyperparameter Tuning

Extensive hyperparameter tuning was performed to optimize the model. We experimented with various parameters including learning rates, batch sizes, and weight decay values. The final parameters that yielded the best performance are:

Hyperparameter	Value
Learning Rate	0.0001
Weight Decay	0.00001
Batch Size	64

Table 2: Final hyperparameters used for training.

4.5 Ablation Studies

To understand the impact of each component in our model, we conducted ablation studies:

- **Removing BERT**: Replacing BERT with simpler text processing models like LSTM resulted in a significant drop in performance, confirming BERT's effectiveness in handling textual data.
- **Simpler CNN**: Using a simpler CNN like ResNet18 instead of ResNet50 also led to a reduction in accuracy, underscoring the importance of using a more powerful CNN for feature extraction.

These experiments highlight the importance of using advanced models like ResNet50 and BERT for multi-label classification tasks, as well as the significance of thorough hyperparameter tuning.

Our experiments clearly show that the combination of ResNet50 and BERT is highly effective for multi-label classification tasks. The model's performance is significantly better than simpler alternatives, and thorough hyperparameter tuning further enhances its accuracy. The results emphasize the value of leveraging powerful models and fine-tuning them appropriately for the task at hand.

5 Conclusions and Discussion

In this study, we successfully implemented a multi-label classification model using ResNet50 for image processing and BERT for text processing. Our model achieved an impressive F1 score of 94.364 on the Kaggle test set, significantly outperforming other tested configurations.

5.1 Key Findings

- **Superior Performance of BERT:** BERT demonstrated its effectiveness in handling text data, significantly outperforming LSTM.
- **Importance of ResNet50:** The use of ResNet50 over simpler CNN models like ResNet18 showed better feature extraction capabilities, leading to higher accuracy.

5.2 Discussion

Our results emphasize the importance of using powerful models for complex tasks. The combination of ResNet50 and BERT leverages the strengths of both image and text processing, resulting in a robust and accurate model. This approach can be applied to various multi-label classification tasks, showcasing the versatility and power of advanced deep learning techniques.

5.3 Future Work

Further improvements can be explored by integrating more advanced architectures and experimenting with different data augmentation techniques. Additionally, fine-tuning the models on specific subsets of the data could lead to even better performance.

Overall, this study highlights the potential of combining state-of-the-art models for multi-label classification, providing a strong foundation for future research and applications in this field.

Ultimately, this study serves as a testament to the enduring relevance of MLPs and provides a roadmap for their optimization in contemporary machine learning tasks. Our approach highlights the importance of methodical experimentation and in-depth understanding of each component's impact on the model's learning and performance.

6 References

[1] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single shot MultiBox detector. In European Conference on Computer Vision (ECCV) (pp. 21-37). Springer, Cham.

[2] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition.

[3] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding.

[4] Alfaro, R., Allende-Cid, H., & Allende, H. (2023). Multilabel text classification with label-dependent representation. Applied Sciences, 13(6), 3594.

[5] Xiao, L., Huang, X., Chen, B., & Jing, L. (2019). Label-specific document representation for multi-label text classification. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP) (pp. 466–475). Hong Kong, China: Association for Computational Linguistics.

[6] Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., Ding, D., Bagul, A., Ball, R. L., Langlotz, C., Shpanskaya, K., Lungren, M. P., & Ng, A. Y. (2017). CheXNet: Radiologist-level pneumonia detection on chest X-rays with deep learning.

[7] Alyoubi, K. H., Alotaibi, F. S., Kumar, A., Gupta, V., & Sharma, A. (2023). A novel multi-layer feature fusion-based BERT-CNN for sentence representation learning and classification. Robotic Intelligence and Automation, 43(6), 704-715.

Appendix

How to Run the Code:

1. Access the code and datasets in the Google Cloud folder at: `Google Drive Link`.
2. Open the notebook `codes.ipynb` in Google Colab.
3. Upload `multi-label-classification-competition-2024.zip` to Colab, then unzip it to the `/content/` directory using the following code:

```
!unzip multi-label-classification-competition-2024.zip -d /content/
```

Alternatively, use the following method to mount and extract from Google Drive:

```
!pip install -q pydrive

from google.colab import drive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

drive.mount('/content/drive/')

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

file_id = '1Y9JpEf_Y22bZYqm6YbmFkorjmG2m_9bo'
file_name = 'multi-label-classification-competition-2024.zip'

download = drive.CreateFile({'id': file_id})
print(f'Downloading {file_name}...')
download.GetContentFile(file_name)
print(f'{file_name} downloaded successfully.')

import zipfile

with zipfile.ZipFile(file_name, 'r') as zip_ref:
zip_ref.extractall('/content/')
print(f'{file_name} extracted successfully.')

!ls -lh /content/
```

4. Ensure all files are completely uploaded before starting the execution of the code.
5. Execute all cells in the notebook by pressing `Ctrl+F9` or by selecting *Runtime* $\rightarrow$ *Run all* from the Colab menu.