Deep Learning Documentation

Monkey_Stack 2024
ISIC 2024 - Skin Cancer Detection with 3D-TBP

Introduction: Why Did We Choose This Project?

Among many intriguing topics, we selected skin cancer detection because we wanted to delve into an image processing task that would deepen our understanding of convolutional neural networks (CNNs). Additionally, the medical imaging domain in AI presented a fascinating and practical opportunity to acquire knowledge applicable in real-life scenarios.

Data Processing

Data processing was one of the most challenging aspects of the task, largely due to the massive dataset involved. With nearly 380,000 images stored in an unbalanced manner within a single HDF5 file, managing this dataset was no small feat. Locally, even unzipping the file proved problematic.

Our solution was to leverage Google resources. We uploaded the compressed file to Google Drive, downloaded it into Google Colab, extracted its contents, and reorganized the images into two separate folders, “Malignant” and “Benign,” based on their IDs. This organization facilitated future tasks. We then recompressed the reorganized files and uploaded them back to Google Drive to streamline subsequent processes.

The dataset's significant issue was its imbalance: approximately 390 malignant images versus hundreds of thousands of benign ones. Such an unbalanced dataset is difficult to work with. To address this, we:

1. Allocated 10% of the malignant and benign images for testing. These test images were intentionally left unbalanced to reflect real-world conditions.
2. Augmented the remaining 350 malignant images to 5,000.
3. Dynamically selected 5,000 benign images for training in each epoch to emphasize generalization.

To load the data into the model, we employed a data generator. This was essential as the dataset could not fit into memory, and the generator provided additional functionality, such as scaling, resizing images, and shuffling the data.

The Image Recognition Model

The image recognition model proved to be a major challenge. We began early with the straightforward idea of building a CNN with a binary classification output, utilizing dense layers. However, this approach encountered several issues. Firstly, we lacked sufficient computational capacity to train a sufficiently large CNN on the entire dataset, especially since we were using free Google Colab resources. Secondly, our dataset at that stage was unbalanced, further complicating the task.

Learning about transfer learning in a lecture opened new possibilities. We hypothesized that by using a pre-trained model, adding a dense layer, and fine-tuning the last 10-20 layers, we could achieve better results without needing enormous computational power. Despite extensive attempts, including many late-night experiments, we failed to achieve accuracy better than random guessing.

We tried several models:

- **EfficientNet-B0**: This model was insufficiently large and lacked accuracy.
- **EfficientNet-V2**: These models were too large to train or optimize within a reasonable timeframe.

This prompted the idea of using balanced datasets, but even that didn't solve the problem.

Inspired by another lecture, we explored **anomaly detection**, which seemed promising for unbalanced datasets. This method involves training the model to reconstruct patterns from the dataset and then using reconstruction quality to determine whether a new image resembles the training data. We quickly implemented a prototype but were disappointed by its lack of significant improvement.

Returning to the basics, we decided to build a custom CNN with **class weights**. Using hyperparameter optimization (similar to a for-loop) to iterate through various weight combinations, we identified suitable values. This approach produced reasonable results—better than random guessing but still unreliable.

Building on this concept, we applied **class weights** to a transfer learning model, reasoning that pre-trained networks inherently possess greater learning capacity. With balanced datasets, transfer learning, class weights, and hyperparameter optimization, we finally developed the best model for the image recognition task so far, but it was not the end.

We ultimately decided on a custom CNN model that is large enough to recognize certain patterns but not so large that running it becomes impractical. With this, we achieved a 69% AUC, meaning we correctly predict 69% of the time. To improve performance, we used class weights, added multiple layers with different sizes, and employed early stopping. The issue with transfer learning was that the pre-trained patterns significantly skewed the predictions in the wrong direction.

It was much easier after we discovered the provided free GPU resources by Kaggle.

We also considered the possibility of training a model on all 380,000 images, with class weight settings that would heavily emphasize the 380 malignant images, with an unbalanced dataset, using an implementation like this:

```
class_weights = compute_class_weight(
  # class_weight='balanced',
)
```

and building a transfer learning model on top of this. However, we quickly abandoned this idea after realizing that even with a small base model, such training would take 50-60 hours for just 30 epochs after some quick calculations."

The Metadata Model

Our efforts did not stop with image recognition. We were confident that better accuracy could be achieved by incorporating **metadata** provided alongside the images on Kaggle. These metadata contained valuable information that we used in a random forest implementation, where we used the output of the CNN image recognition model as a decision point. With this technic we reached a 94 % accuracy from the only 69 % accurate image recognition model.

The metadata, however, presented its own challenges. Certain fields contained missing values, which we handled using various techniques based on the field's relevance:

- Using the average of the available data.
- Omitting the field if it lacked relevance.

Visualization

To aid in understanding and illustrating the task, the **Matplotlib** Python library proved to be immensely helpful. At the beginning of the project, we primarily used Matplotlib for visualizing the data. This allowed us to gain a better sense of the dataset's scale and characteristics by transforming numerical information into human-interpretable images. These visualizations helped us assess the size and content of the data effectively.

Later, Matplotlib played an essential role in other tasks, such as **model evaluation**. The **confusion matrix** became one of the most valuable tools for assessing the model's performance. While there is room for debate about which metric is most important in such tasks, we believed the priority should be to identify **malignant** images as accurately as possible—even if it means mistakenly classifying some benign images as malignant. In a medical context, it is far more critical to catch cancerous cells in time than to avoid unnecessary investigations.

For this reason, two metrics stood out as the most meaningful for us: **Recall** and **AUC (Area Under the Curve)**.

- **Recall** measures the proportion of actual malignant cases correctly identified by the model. It ensures that the model is sensitive to malignant cases, which is crucial in minimizing the chances of missing a diagnosis.
- **AUC** reflects the model's ability to distinguish between classes across all decision thresholds. It provides a comprehensive view of the model's performance, balancing sensitivity and specificity.

These metrics were indispensable in evaluating our model's effectiveness in addressing the critical balance between detecting malignant cases and limiting false negatives.

Conclusion

We believe the final model utilized all available data effectively and provided the best possible predictions based on our knowledge and resources. Throughout this project, we explored and experimented with numerous techniques discussed in lectures, aiming to create the most effective model while gaining practical experience with course materials.

Workflow

Working as a team of three was a highly positive experience. Each member contributed diverse ideas to the project, experimenting with various technologies and sharing their findings. We benefited from our different course schedules, which ensured that someone was always available to work on the project. Task distribution was smooth and fair, with everyone contributing approximately one-third of the work. Being roommates further facilitated quick and effective communication.

The Use of AI

Throughout this project, we frequently relied on **ChatGPT** for assistance. Its most valuable feature was its ability to explain concepts clearly, helping us understand how various metrics work, what they represent, and the purposes of the aforementioned technologies. Gaining this understanding was essential for us to identify the correct tools and methodologies for our task. Typically, we would hear about a concept during a lecture, and ChatGPT would help us contextualize it specifically for our problem. This allowed us to determine whether a particular technology was relevant to solving our challenge.

Additionally, **ChatGPT** and **Gemini** supported us during the coding process. This assistance was particularly important because none of us had worked with Python before this year, and implementing complex logic proved challenging due to our limited familiarity with the required libraries. We used AI to generate code based on the logic we designed, and since Python is a highly readable language, we could easily adapt and personalize the generated code to fit our needs.

We believe the core objective of this course is to learn about **deep learning technologies**, understand their mechanisms, and develop problem-solving skills for related challenges—not to focus solely on Python programming. For this reason, we felt that using AI-generated code enhanced our learning experience rather than detracting from it. It enabled us to explore more technologies and libraries than we could have through documentation alone. Moreover, the time saved by automating parts of the coding process allowed us to focus more on understanding the technologies themselves rather than getting bogged down in implementation details.