Skin Cancer Image Classification From HAM10000 Dataset Using Deep Learning

Group 1

Albert Gu
Muhan An
Ronit Agarwala
Yiqi Liu
Zerlina Lai

Tutorial

In this session, we will go through the entire pipeline of preparing data, fine tuning a model, and evaluating its performance. By the end, you will understand how to use and fine-tune a Vision Transformer model for image classification using Hugging Face's Transformers library.

Introduction

Biopsies, the gold standard to determine whether skin lesions are cancerous, are resource- and time- intensive. Machine learning models have emerged as promising alternatives due to their ability to analyze medical images nearly instantaneously and at low cost [2].

However, training complex models for this task requires a lot of data and compute resources, making them impractical for users to train from scratch. Platforms like Hugging Face aim to solve this problem by making pretrained models available for public use. By leveraging large-scale pretrained models, clinicians can benefit from AI tools that enhance diagnostic precision while reducing unnecessary biopsies.

Aim

The goal of this project is to explore whether pretrained machine learning models can be fine tuned with limited data to accurately classify skin cancer types.

Dataset

HAM10000 [2] is a publicly available dataset of 10015 dermatoscopic images of pigmented skin lesions. It consists of seven diagnostic categories including:

- Basal Cell Carcinoma (bcc)
- Actinic Keratoses and Intraepithelial Carcinoma / Bowen's Disease (akiec)
- Benign Keratosis-like Lesions (bkl)
- Dermatofibroma (df)
- Melanoma (mel)
- Melanocytic Nevi (nv)

- Vascular Lesions (vasc)

Histopathology, expert consensus, follow-up exams, and in-vivo confocal microscopy confirmation are used to confirm the diagnoses in all lesions [1]. The dataset also includes data about the patients' age, sex, and lesion localization (whether it's on the back, scalp, face, etc.)

For this tutorial, due to time and hardware constraints, we select a balanced sample of 100 images from this dataset for the training and testing of our machine learning model.

Step 1: Install and Import Libraries

Description

Before we begin, we need to install and import the necessary Python libraries. These libraries will help us with data loading, model training, and performance evaluation. Some of the most important libraries we work with are:

- Transformers from HuggingFace [3]: Contains important libraries necessary for training and evaluating transformer models from the HuggingFace Hub.
- Scikit-Learn [4], Evaluate: Important for evaluation of different metrics like accuracy, precision, recall, and F1 score.
- Pillow [5]: Adds image processing capabilities to our Python interpreter.
- Torchvision [6]: provides tools for image processing, particularly for deep learning applications using PyTorch.

Step 2: Loading the Dataset

Objective

We will use a dataset of 100 medical images, which we will download from GitHub. This dataset contains labeled images of different skin lesions.

Steps:

1. Clone the dataset repository.
2. Load the image labels into a Pandas DataFrame.
3. Ensure there are no missing images.

About the Data:

- The dataset consists of 100 images with corresponding labels, randomly selected from 10000 original images while ensuring roughly balanced classes
- The labels are stored in a CSV file

- We check for missing files to ensure the dataset is complete.

Step 3: Data Preprocessing

Objective

We need to preprocess our dataset by converting the labels from string format to numerical values, and splitting the data into training and test sets.

Steps:

1. Convert string labels to integer indices.
2. Apply data augmentation to prevent overfitting.
3. Convert the dataset into a format compatible with Hugging Face's Trainer API.
4. Split the dataset into 80% training and 20% test data.

Step 4: Loading and Preparing the Model

Objective

We will load a pre-trained Vision Transformer (ViT) model from Hugging Face and apply image transformations to prepare our dataset for training.

Steps:

1. Load the ViT model (`google/vit-base-patch16-224-in21k`).
2. Define transformations such as cropping, normalization, and tensor conversion.

Step 5: Training the Model

Objective

Now, we will fine-tune the ViT model on our dataset.

A Vision Transformer (ViT) splits an image into small patches and processes them as tokens. ViTs are able to capture both local and global relationships through attention mechanisms, making them more robust than traditional CNN-based architectures.

Steps:

1. We fix a number of hyperparameters before training the model. Some of the most important ones include:

‘num_train_epochs = 30’ : Train the model for 10 epochs over the training dataset.

‘learning_rate = 5e-5’ : Set the initial learning rate, which decides how big a step the model will take during gradient descent initially.

‘per_device_train_batch_size = 16’ : Set the batch size for training. The model will perform training on 16 batches at a time using mini-batch gradient descent.

‘eval_strategy = “epoch”’ : Evaluate the model on the train and test datasets after each epoch.

2. We create a HuggingFace Trainer and load our training and evaluation datasets, training hyperparameters, and our compute_metrics function.
3. Train the model. Training for 30 epochs took around 6-7 minutes on the Google Colab T4 GPU.

Step 6: Evaluating the Model

Objective

Next, we will focus on the evaluation process of our model, which is a critical step in determining whether the model performs well in real-world applications. Use the test set (20% of the data in the sample) to evaluate the generalization of the model on unseen data.

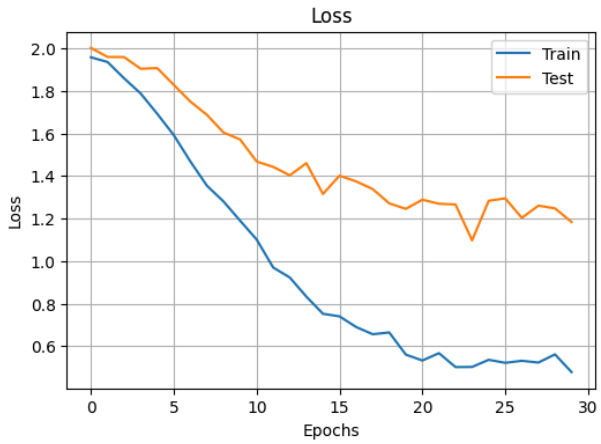
Steps:

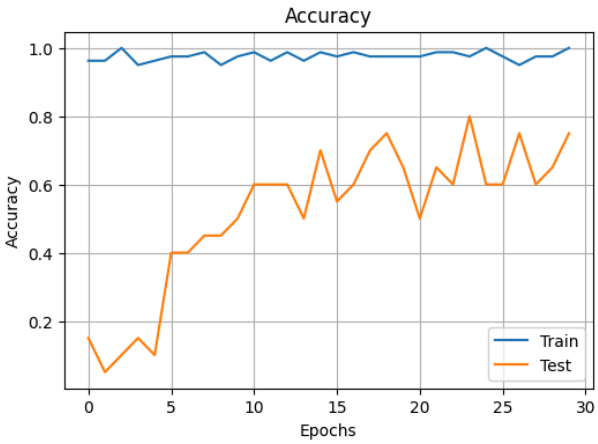
We will evaluate the model based on the following key metrics using `sklearn.metrics`:

1. Accuracy(`accuracy_score()`): Overall proportion of correctly classified images.
2. Precision & Recall & F1-Score(`precision_recall_fscore_support()`): Precision and recall evaluate model’s sensitivity and specificity for each skin condition, while F1-score balances them to provide a comprehensive performance measure.
3. View the trend of loss and accuracy curve: Loss – Quantifies the error at each training step. The lower the loss, the better the model fits the training data.
4. Classification Report (using classification_report())
5. Confusion Matrix(plot heatmap): To analyze misclassification patterns among different lesion types.

Expected Outcome

1. Train Accuracy and Test Accuracy are close and continuously improving & Train Loss and Test Loss are decreasing simultaneously ⟶ the model generalizes well without overfitting and learns effectively

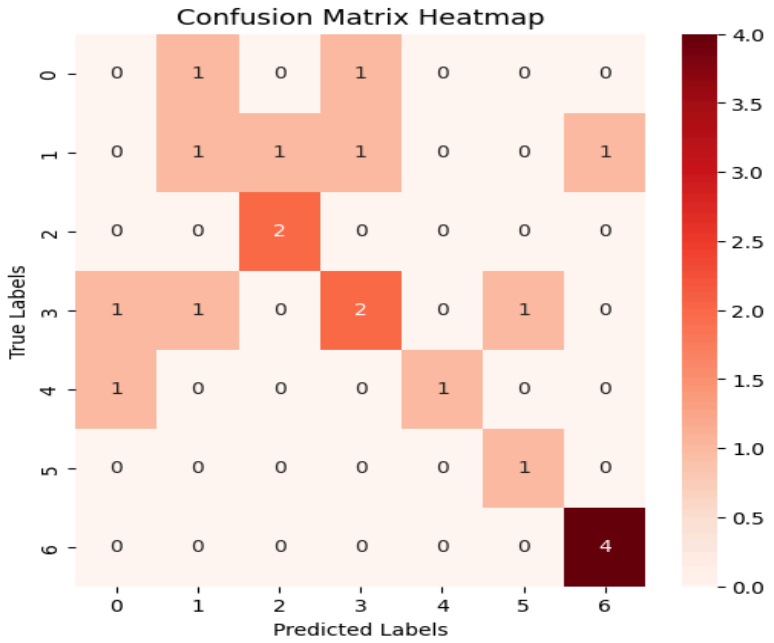




2. Precision, Recall, and F1-score are balanced → model performs well across different classes without bias towards a specific category, minimizing error classifications

	precision	recall	f1-score	support
0	0.00	0.00	0.00	2
1	0.33	0.25	0.29	4
2	0.67	1.00	0.80	2
3	0.50	0.40	0.44	5
4	1.00	0.50	0.67	2
5	0.50	1.00	0.67	1
6	0.80	1.00	0.89	4
accuracy			0.55	20
macro avg	0.54	0.59	0.54	20
weighted avg	0.54	0.55	0.53	20

3. Heat Matrix → model has minimal misclassifications across all lesion types



4. Please note that the exact values for the classification metrics may differ slightly for each run due to the inherent stochasticity present within these algorithms. However, the overall trends of the data should remain the same.

Step 7: Model Optimization

Objective

After evaluating the model's performance, the next step is **optimizing** the model to improve accuracy, reduce loss, and enhance generalization. This step focuses on fine-tuning hyperparameters, addressing overfitting, and improving class-wise performance metrics such as Precision, Recall, and F1-score.

Steps:

1. Expanding the Dataset: A small dataset may cause overfitting and poor generalization. To address this, we can use a larger sample size data.

2. Hyperparameter Tuning: To achieve better performance, we can adjust the key hyperparameter such as Learning Rate(LR). If the model is learning slowly, we can increase the LR. If the model is unstable, we can decrease the LR.

3: Different Model Selection: We can also try training a different model designed to perform better on our small dataset. The model microsoft/swin-tiny-patch4-window7-224 is an example of a transformer model that has linear computational complexity in input image size compared to the quadratic one for our previous model. In theory, this model should train faster on smaller datasets, allowing us to train for more epochs.

Pitfalls to Avoid

1. Dataset Bias: A common issue faced while training machine learning models is inherent bias in the training dataset. In the case of this project we ran into this problem while training our model on a random unbalanced sample from the HAM10000 dataset. Since two-thirds of skin lesions in this dataset belonged to the melanocytic nevi (nv) class, this caused our model to not generalize well to the classes in the dataset. Even though the model might give good training accuracy on such a dataset, this can be misleading as it fails to accurately predict other underrepresented classes. This is why measuring results with a wide variety of metrics (precision, recall, F1, etc.) is important to accurately assess the performance of the model.
2. Improper Hyperparameters: Hyperparameter tuning can be just as important while training such models. Poorly chosen hyperparameters can result in slow training performance, or worse, cause training to fail entirely. For example, setting the learning rate ('learning_rate') too low during training can lead to the model taking too long to converge. Setting it too high, on the other hand, can cause it to fail to converge entirely. Testing out different hyperparameters and monitoring the training and evaluation loss with each epoch is one way to make sure we pick the right hyperparameters.

References

1. Samanta, F. K. (n.d.). Skin cancer dataset. Kaggle. Retrieved February 12, 2025, from https://www.kaggle.com/datasets/farjanakabirsamanta/skin-cancer-dataset/data
2. Tschandl, P., Rosendahl, C., & Kittler, H. (2018). The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Scientific Data, 5, 180161. https://doi.org/10.1038/sdata2018161
3. Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, & Alexander M. Rush (2020). Transformers: State-of-the-Art Natural Language Processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (pp. 38–45). Association for Computational Linguistics.
4. Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, & Gaël Varoquaux (2013). API design for machine learning software: experiences from the scikit-learn project. In ECML PKDD Workshop: Languages for Data Mining and Machine Learning (pp. 108–122).
5. Clark, A. (2015). Pillow (PIL Fork) Documentation. readthedocs. Retrieved from https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf

6. Maintainers, T., & Contributors. (2016). TorchVision: PyTorch's Computer Vision library. Retrieved from https://github.com/pytorch/vision
7. Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby (2020).