Accelerating Image Classification Using Transfer Learning in PyTorch

Akhilesh Rajendra Shinde
SYMCA
PES Modern College of Engineering,
Pune-5

Prof. Y.L. Puranik
Assistant Professor,
MCA Department,
PES Modern College of Engineering,
Pune-5

Abstract:

Transfer Learning for Image Classification in PyTorch Transfer learning is a powerful technique for training deep learning models, especially for image classification tasks, when datasets are limited. This approach leverages a pre-trained model on a large dataset like ImageNet, and then adapts it to a new, smaller dataset for a specific classification task. extend this no extra heading Show drafts While transfer learning excels in overcoming limited datasets for image classification in PyTorch, its benefits extend further. Pre-trained models on massive datasets like ImageNet capture generic image features, acting as a strong foundation for new tasks. This translates to: Faster Training: By leveraging pre-trained weights, especially for early layers, training becomes significantly faster compared to training from scratch. Improved Performance: Fine-tuning pre-trained models on smaller datasets often leads to superior performance compared to training a new model entirely. This reduces the risk of overfitting and leverages the power of learned features. Efficient Model Development: Transfer learning allows developing image classification models for specific tasks even with limited data. This frees researchers from training from scratch, accelerating the development process. PyTorch offers a user-friendly environment for implementing transfer learning: Pre-trained Model Selection: Choose a pre-trained model (ResNet, VGG, EfficientNet) from PyTorch's torchvision library based on factors like dataset size and desired accuracy. Feature Extraction & Fine-tuning: Freeze the pre-trained model's feature extractor (early layers) to retain generic features. Replace the classifier (final layers) with new ones specific to the new dataset's number of classes. Train these new layers on the smaller dataset. Fine-tuning Strategies: Depending on dataset complexity, techniques like unfreezing a few layers closer to the classifier can be used for further performance gains. Beyond the basics, data augmentation

(artificially increasing data size and diversity) and hyperparameter tuning (optimizing learning rate, batch size) can

Introduction

Transfer learning has revolutionized the landscape of image classification in deep learning by enabling the reuse of knowledge gained from previously trained models on large-scale datasets such as ImageNet. Instead of building models from scratch, which often demands vast computational resources and massive labeled datasets, transfer learning allows researchers and practitioners to fine-tune pre-trained models on specific tasks with significantly smaller datasets. This not only cuts down training time substantially but also improves generalization and accuracy, especially in scenarios where labeled data is scarce.

By leveraging deep convolutional neural networks (CNNs) that have already learned rich feature representations, transfer learning

Literature Survey

[1]:Yann LeCun ”Deep Learning for Image Classification" (LeCun et al., 2015), Deep learning has revolutionized image classification by enabling models to automatically learn hierarchical features from raw data, rather than relying on hand-crafted features. Convolutional Neural Networks (CNNs), inspired by the human visual cortex, have proven to be particularly effective for this task. By using layers of convolutional filters, pooling, and nonlinear activations, CNNs can capture spatial hierarchies in images, from edges and textures in early layers to more complex patterns like object parts in deeper layers. This architecture has significantly improved accuracy in tasks such as object recognition, face detection, and scene labeling.

mitigates the risk of overfitting and accelerates convergence during

Further enhance transfer learning effectiveness

training. It has become a standard approach in computer vision applications, offering a practical solution to real-world problems ranging from medical imaging to autonomous driving.

This research focuses on the implementation of transfer learning in **PyTorch**, an open-source deep learning framework that provides dynamic computation graphs and ease of use. We explore how different pre-trained models, including ResNet, VGG, and EfficientNet, can be adapted to specific image classification tasks. The goal is to demonstrate how transfer learning can not only enhance performance but also make image classification more accessible and efficient for domains with limited computational and data resources.

[2]:MaithraRaghu,SamyBengio"Understanding Transfer Learning for Medical Imaging" (Raghu et al., 2019) , The paper investigates the effectiveness of transfer learning from large-scale natural image datasets (like ImageNet) to medical imaging tasks, which are often quite different in content and structure. Surprisingly, the authors found that **transfer learning does not always provide significant performance improvements** in medical imaging, especially when the target datasets are not too small. Shallow models trained from scratch often performed competitively or even better than fine-tuned deep models, suggesting that the standard approach of reusing deep pre-trained features may not be optimal in specialized domains like radiology or pathology.

[3] Adam Paszke,Sam Gross ,"PyTorch: An Imperative Style, High-Performance Deep Learning Library" (Paszke et al., 2019),PyTorch is built around an imperative programming style, which means computations are executed as they are written, unlike static graph-based systems where the entire computation graph must be defined before execution. This dynamic approach allows for greater flexibility and easier debugging, making it especially appealing to researchers. The paper emphasizes how PyTorch blends deep integration with Python and supports native control flow, making model development intuitive

[4] Pan & Young,"A Comprehensive Review on TransferLearning for Image Classification" (Pan & Yang, 2010): Transfer learning has emerged as a vital technique in image classification, enabling models to leverage knowledge gained from one domain and apply it to another. This approach is particularly beneficial when the target dataset is limited or expensive to label. The paper reviews various strategies including feature extraction, fine-tuning, and domain adaptation. It also explores how models pre-trained on large datasets like ImageNet can be adapted to more specialized tasks such as satellite image recognition, medical diagnosis, and facial emotion detection.

[5] Howard & Ruder,"Transfer Learning from Pre-trained Models" (Howard & Ruder, 2018),Howard and Ruder introduced **ULMFiT**, a transfer learning method that adapts pre-trained language models to text classification tasks with remarkable efficiency and accuracy. The method involves three stages: (1) pre-training a language model on a large general corpus (like Wikipedia), (2) fine-tuning the model on a target task corpus, and (3) training a classifier on the adapted model. This technique significantly reduces the need for large labeled datasets in the target domain, allowing models to generalize well even with limited data.

[6] Zhu,"Transfer Learning in Convolutional Neural Networks for Computer-Aided Detection of Mammographic Masses" (Zhu et al., 2016) The paper investigates how transfer learning can be used to overcome the limitations of small annotated datasets in medical imaging, especially for mammographic mass detection. By employing convolutional neural networks pre-trained on large natural image datasets like ImageNet, and fine-tuning them on mammogram images, the authors demonstrate improved classification performance compared to training CNNs from scratch. This method allows the network to retain low-level visual features such as edges and textures while learning domain-specific patterns related to tumor appearance.

Proposed Methodology

- **Integrating Dataset Downloading into Code**

Deep learning libraries like PyTorch offer functions such as download_url to automate dataset downloading within code. This function takes a URL and destination as arguments, downloading the dataset from the specified URL to the designated location on your local machine. Benefits include convenience, integration with code, and ensuring reproducibility of experiments.

Fig1:Dataet download

- **Creating a Custom PyTorch Dataset**

Creating a custom PyTorch dataset involves defining a class that inherits from torch.utils.data.Dataset and implementing methods like `__len__` to return the total number of samples and `_getitem__` to load and preprocess specific data samples. This method encapsulates dataset handling logic, promoting code organization and clarity. Additionally, leveraging existing data loading utilities for common operations like image resizing or random cropping from `torchvision.transforms` can streamline dataset augmentation. This approach offers flexibility in controlling data loading and transformation while promoting reusability across different projects.

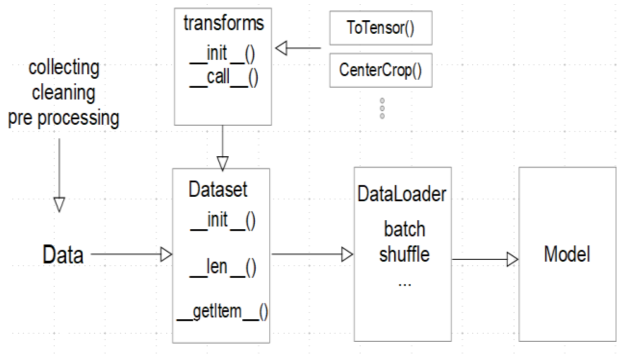


Fig 2:Custom PyTorch Dataset

- **Creating Training and Validation Sets:**

Splitting data into training and validation sets is crucial in machine learning, particularly for preventing overfitting. Common strategies include random split, dividing the dataset into training (70-80%) and validation (20-30%) sets, and stratified split, which maintains class proportions, useful for imbalanced datasets. Validation sets allow for model evaluation, hyperparameter tuning, and early stopping to

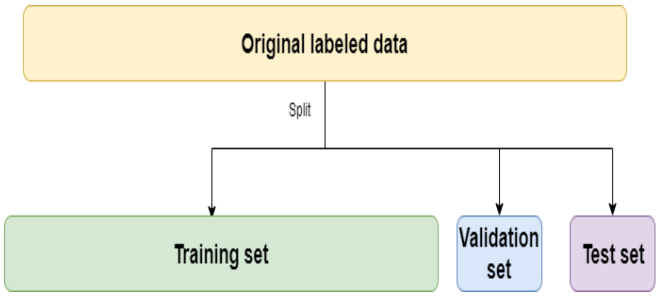
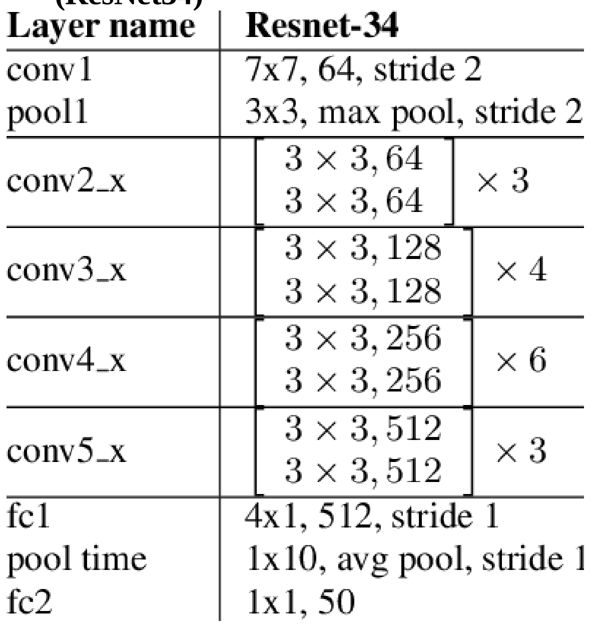


Fig 3:Training and Validation

prevent overfitting by monitoring performance on unseen data. This process enhances generalization and ensures optimal model performance.

- **Modifying a Pretrained Model (ResNet34)**



Layer name	Resnet-34
conv1	7x7, 64, stride 2
pool1	3x3, max pool, stride 2
conv2_x	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$
conv3_x	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$
conv4_x	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$
conv5_x	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$
fc1	4x1, 512, stride 1
pool time	1x10, avg pool, stride 1
fc2	1x1, 50

Fig 4:Analysis

To modify a pre-trained ResNet-34 model for transfer learning, you can employ either feature extraction or fine-tuning techniques. In feature extraction, you freeze most layers of the pre-trained model, treating them as fixed feature extractors, while adding a new classifier head for task-specific learning. This approach is efficient for smaller datasets and prevents overfitting. Conversely, fine-tuning involves unfreezing a few final layers, allowing the model to adapt pre-learned features to the new dataset through end-to-end training. Fine-tuning may yield higher performance, especially for larger datasets or tasks with significant domain differences. Your choice depends on factors like dataset size, task similarity, and available computational resources.

- **GPU Utilities and Training Loop**

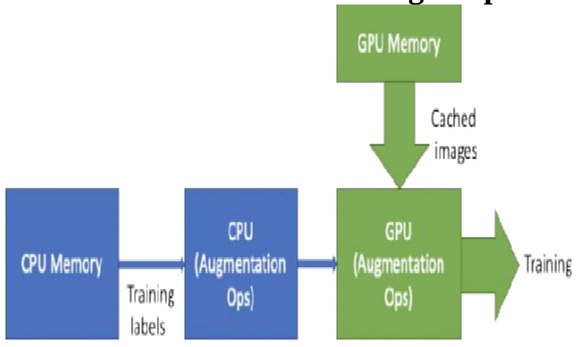


Fig 5:GPU Utilities

GPUs are vital for deep learning training due to their ability to handle parallel computations efficiently. PyTorch offers utilities like torch.device to specify GPU usage and methods like model.to(device) for moving models and data tensors to GPUs. In the training loop, data loading, forward pass, loss calculation, and optimizer update steps are executed on GPUs for faster training. GPU training offers significant speedups over CPUs, enabling efficient training of larger models and handling of complex datasets.

- **Fine Tuning the Pretrained Model**

Fine-tuning a pre-trained model like ResNet-50 for transfer learning involves adapting it to a new, smaller dataset and task. Initially, the pre-trained model is loaded, and its final layers are modified to suit the new classification task, typically by adding a new classifier head. While most layers are frozen, a few final convolutional layers might be partially unfrozen to adapt to task-specific features. Training the model with an optimizer and loss function allows the frozen layers to act as feature extractors while the new layers learn task-specific features. Fine-tuning offers faster training and potentially improved performance, especially for smaller datasets, but its effectiveness depends on the similarity between the new and original tasks.

Implementation

- **Downloading a Dataset:**

The code utilizes the download_url function from torchvision.datasets.utils to simplify dataset downloading, specifying the dataset's URL (https://s3.amazonaws.com/fast-ai-imageclas/oxford-iiit-pet.tgz) and destination (the current working directory). Upon execution, download_url establishes a connection to the URL, retrieves the dataset file, and saves it at the designated location. Additionally, the code imports the tarfile module to handle compressed archive files and extracts the downloaded ".oxford-iiit-pet.tgz" file, placing its contents within a new "data" directory in the current working directory.

- **Creating a Custom PyTorch Dataset**

The code defines a class named `PetsDataset` that inherits from `torch.utils.data.Dataset`, designed for handling a dataset containing pet images. It includes initialization to store dataset root and transformation objects, along with methods to get dataset length and access individual samples. The dataset instance is created with transformations for image resizing, padding, cropping, conversion to tensors, and normalization. Additional helper functions are provided for denormalization and displaying images. However, it mentions a missing `parse_breed` function to extract breed labels from filenames.

- **Creating Training and Validation Sets**

The code for splitting the dataset imports the `random_split` function from `torch.utils.data` and defines the validation data percentage `(val_pct)`. It calculates the validation size `(val_size)` based on the total dataset size and uses `random_split` to split the dataset into training and validation datasets `(train_ds and valid_ds)`. The code then creates DataLoaders using the `DataLoader` class from `torch.utils.data`, specifying parameters such as batch size, shuffle, number of workers, and pin memory for both training and validation DataLoaders. Lastly, it defines a function `(show_batch)` to visualize a batch of images from the DataLoader using `make_grid` from `torchvision.utils` and Matplotlib for plotting, denormalizing the images and displaying them in a grid format.

- **Modifying a Pretrained Model (ResNet34)**

The code defines an accuracy function that calculates accuracy by comparing predicted labels with true labels using torch.max. It also introduces an ImageClassificationBase class serving as a base for image classification models, containing methods for training, validation, and evaluation steps. Additionally, the PetsModel class inherits from ImageClassificationBase and loads a pretrained ResNet34 model, replacing its final fully connected layer with a new one for the given task. The forward pass simply passes input data through the loaded ResNet34 model.

- **Fine Tuning the Pretrained Model**

The process starts by creating a PetsModel instance with the number of output classes matching the dataset's unique breeds and moving it to the appropriate device using `to_device`. Then, an initial evaluation of the untrained model on the validation set is conducted using

- **GPU Utilities and Training Loop**

The code snippet introduces GPU utilities and a training loop for deep learning models. It includes functions for managing device selection, training, and learning rate scheduling. The fit function trains a model for a specified number of epochs, while `fit_one_cycle` implements a one-cycle learning rate scheduler. The snippet also demonstrates device usage by wrapping data loaders in `DeviceDataLoader` instances to ensure automatic data transfer to the chosen device during training.

`model.evaluate(valid_dl)`, with results stored in a list named `history`. Next, training parameters such as the number of epochs, maximum learning rate, gradient clipping value, weight decay, and optimizer function are defined. Finally, training with a one-cycle scheduler is executed using `fit_one_cycle`, passing the specified parameters. The %%time magic command estimates the execution time of the training process.

Future Scope

Model Improvements: Experiment with different pre-trained CNN architectures like EfficientNet, DenseNet, or VGG models for potential performance enhancements. Consider fine-tuning pre-trained models by freezing earlier layers and training only specific final layers tailored to your pet classification task. Employ data augmentation techniques such as random cropping, flipping, rotation, and color jittering to augment training data diversity, potentially enhancing model robustness. Leverage transfer learning by pre-training models on similar classification tasks with larger datasets before fine-tuning them on your pet image dataset, especially beneficial for smaller datasets. Advanced Training Techniques: Explore various learning rate scheduling techniques like cosine annealing or cyclical learning rates for potential performance improvements.

Conclusion

Functionality: Defines a custom dataset class (PetsDataset) to handle pet image data. Implements helper functions for data management, device usage, training, and evaluation. Creates a PetsModel that leverages a pre-trained ResNet34 architecture for pet breed classification. Trains the model using a one-cycle learning rate scheduler for efficient optimization.
Strengths: Modular and well-organized code structure. Leverages pre-trained models for efficient learning. Employs a modern learning rate scheduler for training. Includes functionalities for data management, device handling, and evaluation.

Future Enhancements: Experiment with different model architectures and training techniques. Consider data augmentation and transfer learning if applicable. Explore advanced evaluation metrics and visualization tools. Deploy the trained model for real-world applications.

Reference

[1] Kayla Mendel, Huili Wang, Daniel Sheth, Maryellen Giger.Transfer Learning From Convolutional Neural Networks for Computer-Aided Diagnosis: A Comparison of Digital Breast Tomosynthesis and Full-Field Digital Mammography. This study compares the performance of deep learning CAD systems on Digital Breast Tomosynthesis (DBT) and Full-Field Digital Mammography (FFDM) images. Utilizing a pre-trained CNN for feature extraction, the researchers found that DBT images provided superior diagnostic performance over FFDM, particularly in classifying masses and architectural distortions. This suggests that DBT may offer more relevant information for lesion malignancy assessment when analyzed with CNN-based CAD systems.

[2] Ravi K. Samala, Heang-Ping Chan, This paper introduces a multi-task transfer learning approach using deep CNNs to classify malignant and benign breast masses. By training the network on both screen-film and digital mammograms, the model demonstrated improved generalization and diagnostic accuracy compared to single-task learning. The study highlights the effectiveness of multi-task transfer learning

in enhancing CAD systems for breast cancer detection

[3] Aditya Khamparia, Subrato Bharati. Diagnosis of Breast Cancer Based on Modern Mammography Using Hybrid Transfer Learning. The authors propose a hybrid transfer learning model combining modified VGG16 and ImageNet architectures to improve breast cancer diagnosis from mammograms. Evaluated on the DDSM dataset, the hybrid model achieved an accuracy of 88.3%, outperforming individual architectures. This approach demonstrates the potential of combining multiple pre-trained models to enhance CAD performance in mammography.

[4] Li Shen, Laurie R. Margolies. Deep Learning to Improve Breast Cancer Early Detection on Screening Mammography. This study presents a deep learning algorithm trained to detect breast cancer on screening mammograms using an end-to-end approach. The model achieved high accuracy on both DDSM and INbreast datasets, with area under the curve (AUC) values of 0.91 and 0.98, respectively. The research demonstrates the feasibility of applying deep learning to improve early breast cancer detection in clinical settings.

[5] Hang Min, Devin Wilson. Fully Automatic Computer-Aided Mass Detection and Segmentation via Pseudo-Color Mammograms and Mask R-CNN. The researchers developed a fully automatic CAD system that employs pseudo-color mammograms and Mask R-CNN for simultaneous mass detection and segmentation. By enhancing grayscale mammograms into pseudo-color images, the model improved the performance of mass detection and segmentation tasks. Evaluated on the INbreast dataset, the system achieved a true positive rate of 0.90 and a Dice similarity index of 0.88, indicating high accuracy in identifying and delineating breast masses.

[6]S.R.SannasiChakravarthy,N.Bharanidharan. Transfer Learning for Deep Neural Networks-Based Classification of Mammograms. This study proposes a deep learning model combining transfer learning and long short-term memory (LSTM) networks to enhance breast mass detection and diagnosis. The model aims to improve therapy outcomes and reduce mortality risks by accurately identifying suspicious regions in mammogram images.

[7] Daniel G. P. Petrini, Carlos Shimizu. Breast Cancer Diagnosis in Two-View Mammography Using End-to-End Trained EfficientNet-Based Convolutional Network. This paper presents a deep learning model using EfficientNet for breast cancer diagnosis in two-view mammography. The model employs multiple transfer learning stages to classify mammograms, achieving high accuracy and sensitivity in detecting breast cancer