Member-only story

Transfer Learning For Beginner

A practical guide to transfer learning in image classification

Mina Ghashami · Follow

Published in Towards Data Science · 7 min read · Oct 29, 2023

In this post, we will look at the concept of *transfer learning*, and we will see an example of it in *image classification task*.

What is Transfer Learning?

> *Transfer learning is a technique in deep learning where pre-trained models trained on large-scale datasets are used to solve new tasks with limited labeled data.*

It involves taking a pre-trained model, which has learned rich and generalized feature representations from a source task, and fine-tuning it on a target task.

For example, ImageNet which is a large dataset (14 million images of 1000 classes) are often used to train large convolutional neural networks such as VGGNet or ResNet.

If we train these networks are ImageNet, these models learn to extract powerful and informative features. We call this training *pre-training* and these models are *pre-trained on ImageNet*. Note they are trained for image classification task on ImageNet. We call it the *source task*.

To do transfer learning on a new task which we call *target task*, first of all we need to have our labelled dataset which is called *target dataset*. Target dataset is often much much smaller than source dataset. Our source dataset here was huge (it has 14 million images).

Then, we take these pre-trained models and chop off the final classification layer, and add a new classifier layer at the end and train them on our own target dataset. When we are training, we freeze all layers except the last layer, as a result very few parameters are getting trained and therefore training happens fast. And Voila! we have done transfer learning.

The second training that model goes through is called *fine-tuning*. As we saw, during fine-tuning, most of the pre-trained weights are frozen, and only the final layers are adjusted to the new dataset.

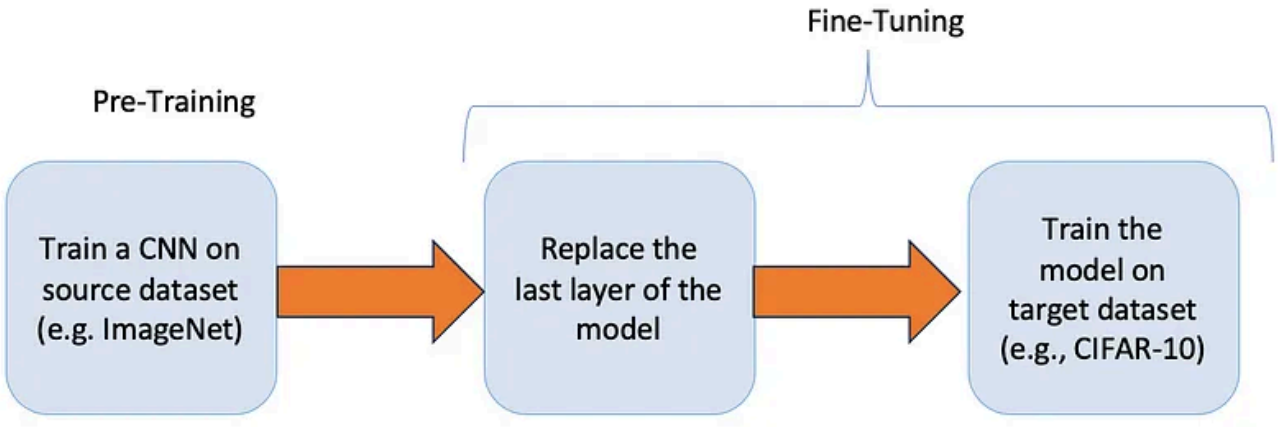


Image by the author

Benefits of Transfer Learning

The key advantages of transfer learning are that it allows you to capitalize on the expertise already developed in pre-trained models, hence avoids training large models from scratch. It also mitigates the need for large labeled datasets which are time-consuming to collect and annotate.

Fine-tuning a pre-trained model is much faster and computationally cheaper than training from scratch. These models often achieve high accuracy by building on top of general features learned during pretraining.

Caveats of Transfer Learning

Caveats of transfer learning is that the target task and dataset has to be close to the source task and dataset. Otherwise the knowledge learned during pre-training will be useless for the target task. If that's the case, we are better off training the model from scratch.

Practical Example

We will use VGGNet to demonstrate transfer learning. In a previous post, we looked at VGGNet. Take a look if you are unfamiliar.

VGG (Visual Geometry Group) is a deep convolutional neural network (CNN) architecture developed by the Visual Geometry Group at the University of Oxford. It comes in many variants such as VGG16 and VGG19. All variants

have similar architecture except the number of layers are different. For example, VGG-16 has 16 layers, including 13 convolutional layers and 3 fully connected layers.

VGGNet trained on ImageNet is often used as a pre-trained model for transfer learning in image classification.

For fine-tuning, we will use STL10 datasets which has 5000 small-sized color images (96x96 pixels) from 10 different classes. The dataset is split into a training set of 5000 images and a test set of 8000 images.

The STL10 dataset is our target dataset and the ImageNet is our source dataset and they are very similar in nature so it makes sense to use them in transfer learning.

Here is a summary of our setup in a table:

Source Task	Image Classification
Source Dataset	ImageNet
Target Task	Image Classification
Target Dataset	STL10
Pre-trained Model	VGG16
Fine-tuned Model	Chop off last layer of VGG16, add a new classification head. Freeze all previous layers' parameters during fine-tuning.

image by author

Load The Pretrained Model

Since the last layer of VGG is 1000 (as it was trained for ImageNet which contains 1000 classes) we are removing that replacing it with a layer of 10 classes.

```
from torchvision.models import vgg16

# Load the pre-trained VGG-16 model
vgg = vgg16(pretrained=True)
print(vgg)
```

When we print VGG architecture, we see the following: the last 3 layers are the fully connected layers, of which the last fully connected layer is the classification head that classifies an input of 4096 dimension into 1000 classes.

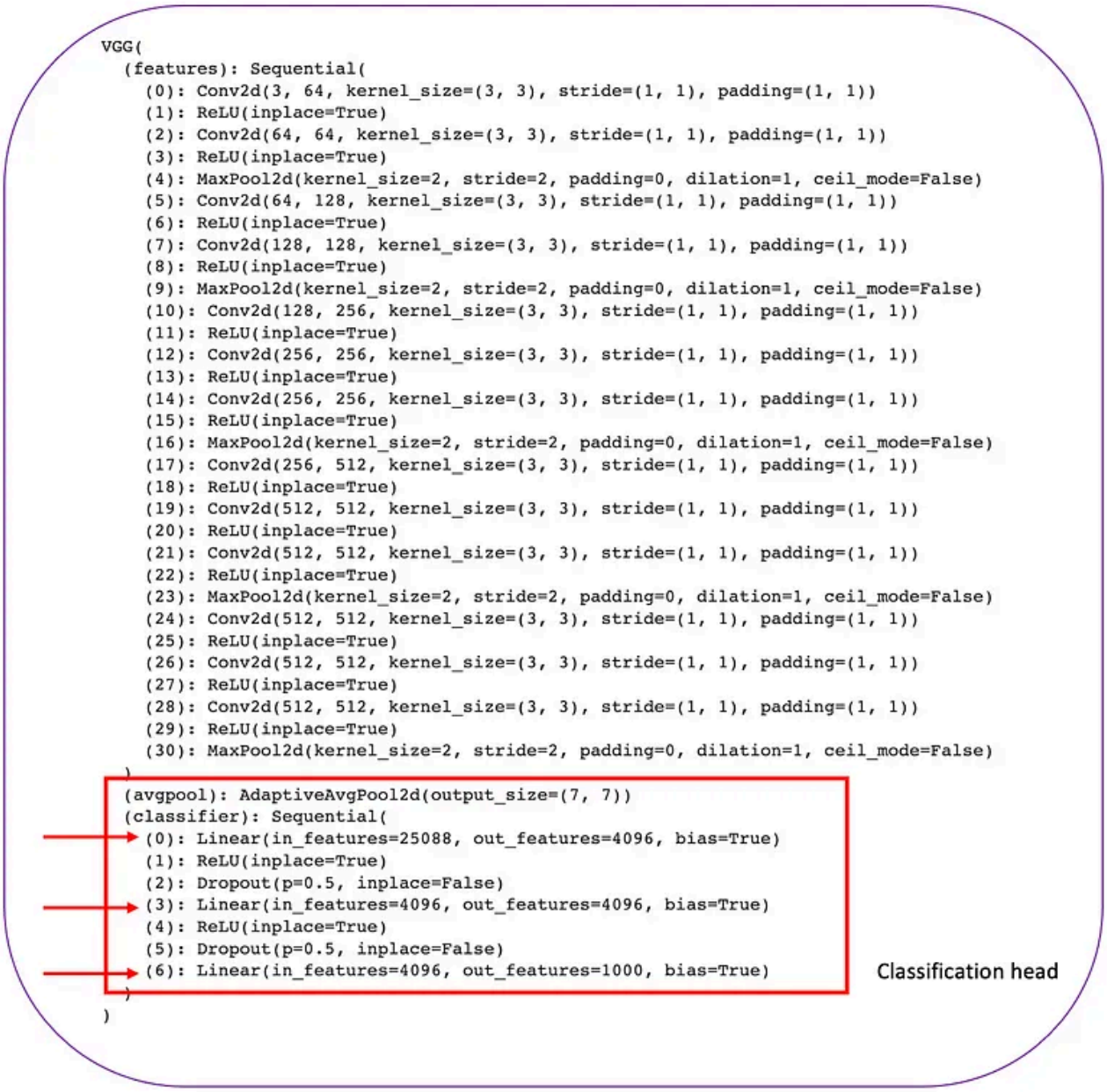


Image by author

We need to chop off last layer and put a new layer that classifies input into 10 classes! because STL10 has only 10 classes. So we do:

```
# Modify the last layer of VGG by changing it to 10 classes
vgg.classifier[6] = nn.Linear(in_features=4096, out_features=len(classes))
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu");
vgg.to(device)
```

Data Preparation

We first load and transform our target data. The code is as following:

```
# train transformation
transform_train = transforms.Compose([
    transforms.RandomCrop(96, padding = 4), # we first pad by 4 pixels on each s
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.44671103, 0.43980882, 0.40664575), (0.2603408 , 0.25
])

# test transformation
transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.44671103, 0.43980882, 0.40664575), (0.2603408 , 0.25
])

trainset = torchvision.datasets.STL10(root = './data', split = 'train', download
trainloader = torch.utils.data.DataLoader(trainset, batch_size = 128, shuffle =

testset = torchvision.datasets.STL10(root = './data', split = 'test', download =
testloader = torch.utils.data.DataLoader(testset, batch_size = 256, shuffle = Tr
```

We explain each section. First,

```
transforms.RandomCrop(96, padding = 4)
transforms.RandomHorizontalFlip(),
```

RandomCrop() takes two arguments — output size and padding. For example, with output size 32 and padding 4, it first pads the image by 4 pixels on each side, then takes a random 32x32 crop from the padded image.

This allows the crop to include pixels from the edges of the original image, hence provides data augmentation by generating diverse crops from the same input. Without padding, the crops would always be from the center and not include edge regions.

This data augmentation, helps expose the model to different parts of the image, and improves generalization.

Second,

```
transforms.ToTensor()
```

The *ToTensor()* converts a PIL image or numpy array to a Tensor that can be fed into a neural network. It handles all the transformations required to go from image data to PyTorch-compatible tensor such as normalizing the data so that they are in (0,1) range, and transposes (H, W, C) array to (C, H, W) for PyTorch model input. For example, a RGB image would become a 3xHxW Tensor, and a grayscale image becomes a 1xHxW Tensor.

Lastly,

```
transforms.Normalize((0.44671103, 0.43980882, 0.40664575), (0.2603408 , 0.256577
```

normalizes the data by subtracting mean and dividing by standard deviation.

FinetuneThe Model

There are two ways to fine-tune the model:

1. either we freeze the previous layers and let the classification head only be trained.
2. or we train all the layers together.

While the first method is faster, the second will likely be more accurate. Let's first train the model via option 2. For that we need the following functions:

```
def train_batch(epoch, model, optimizer):
    print("epoch ", epoch)
    model.train()
    train_loss = 0
    correct = 0
    total = 0

    for batch_idx, (input, targets) in enumerate(trainloader):
        inputs, targets = input.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs, _ = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += predicted.eq(targets).sum().item()
```

```
        print(batch_idx, len(trainloader), 'Loss: %.3f | Acc: %.3f%% (%d/%d)'
                             % (train_loss/(batch_idx+1), 100.*correct/total, correc


    def validate_batch(epoch, model):
        model.eval()
        test_loss = 0
        correct = 0
        total = 0
        with torch.no_grad():
            for batch_idx, (inputs, targets) in enumerate(testloader):
                inputs, targets = inputs.to(device), targets.to(device)
                outputs,_ = model(inputs)
                loss = criterion(outputs, targets)

                test_loss += loss.item()
                _, predicted = outputs.max(1)
                total += targets.size(0)
                correct += predicted.eq(targets).sum().item()

        print(batch_idx, len(testloader), 'Loss: %.3f | Acc: %.3f%% (%d/%d)'
                     % (test_loss/(batch_idx+1), 100.*correct/total, correct, total)
```

and then putting them together brings us to the full training:

```
    start_epoch = 0
    for epoch in range(start_epoch, start_epoch+20):
        train_batch(epoch, vgg_model, vgg_optimizer)
        validate_batch(epoch, vgg_model)
        vgg_scheduler.step()
```

If we decide to freeze some layers and not train them, we set `requires_grad = False` on the weights and biases of a layer to freeze that layer.

This concludes our topic of transfer learning in image classification.

Conclusion

Transfer learning is a technique where a model trained on one task is reused as the starting point for a model on a second related task. It allows you to leverage knowledge from a pretrained model instead of training a model from scratch. For example, wecan take an ImageNet pretrained model and retrain it on a new dataset of similar images. Features learned by the pre-trained model on the first task are transferred and reused on the new task.

Let me know if you have any comment or questions.

If you have any questions or suggestions, feel free to reach out to me:
Email: mina.ghashami@gmail.com
LinkedIn: https://www.linkedin.com/in/minaghashami/

Deep Learning | Transfer Learning | AI | Convolutional Network | Machine Learning

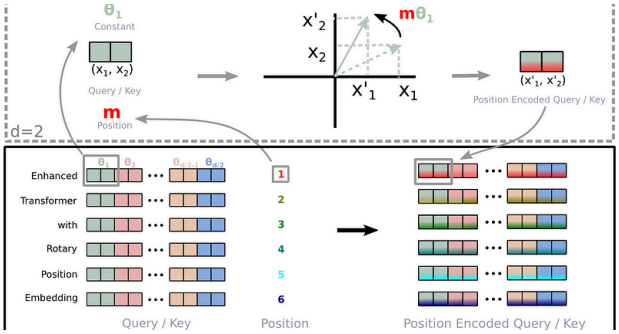
Written by Mina Ghashami

Follow

845 Followers · Writer for Towards Data Science

Applied Scientist @Amazon AWS | Adjunct lecturer@NYU | Previously, Adjunct lecturer@Stanford

More from Mina Ghashami and Towards Data Science



Mina Ghashami in Towards Data Science

Understanding Positional Embeddings in Transformers: Fro...

A deep dive into absolute, relative, and rotary positional embeddings with code examples

Jul 19 230

Pierre-Etienne Toulemon... in Towards Data Scien...

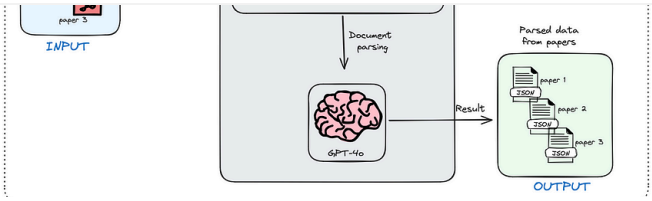
Full Guide to Building a Professional Portfolio with Pytho...

This article is a end-to-end guide to build a professional portfolio for developpers and...

Jul 20 731 4

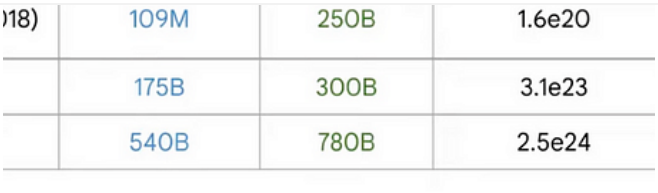
Open in app

Medium Search Write



Zoumana Keita in Towards Data Science

Document Parsing Using Large Language Models—With Code



)18)	109M	250B	1.6e20
	175B	300B	3.1e23
	540B	780B	2.5e24

Mina Ghashami in Towards Data Science

Scaling Law Of Language Models

How language models scale with model size, training data, and training compute

You will not think about using Regular Expressions anymore.

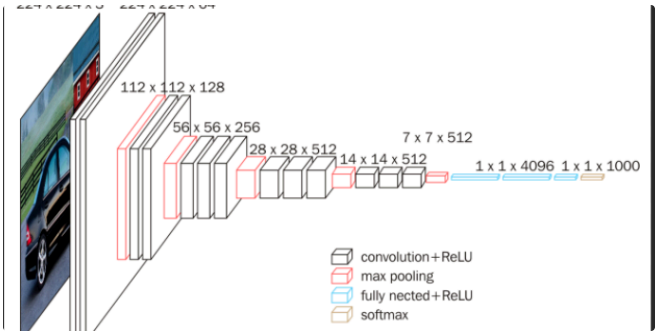
Jul 25 717 5

Jul 9 84 1

See all from Mina Ghashami

See all from Towards Data Science

Recommended from Medium

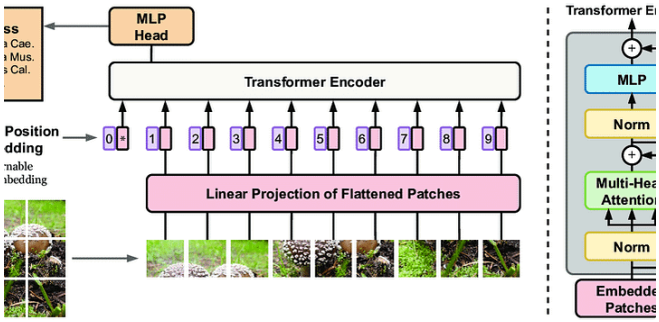


David Fagbuyiro

Guide To Transfer Learning in Deep Learning

In this guide, we will cover what transfer learning is, and the main approaches to...

Apr 20 83



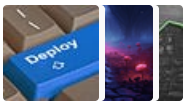
Ankit kumar

Vision Transformer Part 1

— Basics of ViTs (Vision Transformers)

Mar 18 138

Lists

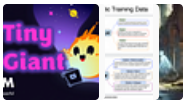


Predictive Modeling w/ Python

20 stories · 1435 saves

Practical Guides to Machine Learning

10 stories · 1744 saves



Natural Language Processing

1640 stories · 1199 saves

The New Chatbots: ChatGPT, Bard, and Beyond

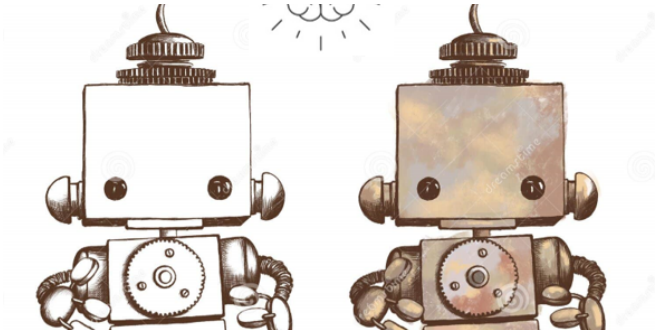
12 stories · 438 saves

Rayyan Shaikh

Mastering BERT: A Comprehensive Guide from Beginner to Advanced...

Introduction: A Guide to Unlocking BERT: From Beginner to Expert

Aug 26, 2023 · 2.1K · 19

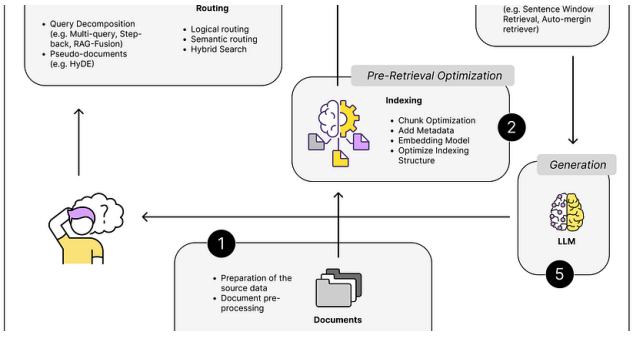


Alessandro Bosco in Data Reply IT | DataTech

Transfer Learning: Feature Extraction and Fine tuning

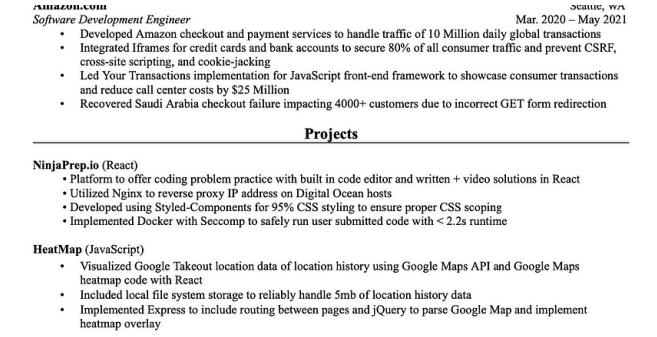
What if we do not have sufficient data for training our model? A solution is Transfer...

Mar 26 · 12



Dominik Polzer in Towards Data Science

17 (Advanced) RAG Techniques to Turn Your LLM App Prototype into...



Software Development Engineer Mar. 2020 – May 2021

- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by $25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

Projects

NinjaPrep.io (React)
- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

HeatMap (JavaScript)
- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay

Alexander Nguyen in Level Up Coding

The resume that got a software engineer a $300,000 job at Google.

1-page. Well-formatted.

A collection of RAG techniques to help you develop your RAG app into something robus...

Jun 26 2.3K 22

Jun 1 17.3K 273

See more recommendations