

Image Classification



Image Classification Project



Submitted to:

Eng. Abdelrahman Shaker

Sprints Tutors

Submitted by:

Mohammad Hesham

Mohammad Tarek

Mohammed El-Sayed

Zahra Abu-Zeid

Nour Gamal

Contents

Introduction:	3
What is CNN:	3
Defining our problem:	3
Dataset:	3
Implementation:	3
Models:	4
The code steps	4
Analysis and opting the best model:	7
Conclusion:	10
References	12

Introduction:

What is CNN:

Image classification is made up of Convolutional neural network (CNN). CNN is a class of deep, feed-forward artificial neural networks, most commonly applied to analyzing visual imagery. CNNs, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output .

So what the difference between the neural network and the CNN ? Unlike neural networks, where the input is a vector, here the input is a multi-channeled image (3 channeled in this case).

What is Image Classification:

Image classification is a supervised learning problem: define a set of target classes (objects to identify in images), and train a model to recognize them using labeled example photos. Early computer vision models relied on raw pixel data as the input to the model.

Defining our problem:

Our Images Dataset needed to be classified by machine learning program, specifically deep learning. Classification contains six categories. After training phase, program should punctually predict the input images' categories.

Categories are: Buildings, forest, glacier, mountain, sea and street

Dataset:

We have unstructured dataset¹ which combined 17084 images. The dataset divided into two sects: Training (14034 image) and Testing (3050 image)

The dataset itself do have some irrelevant sample in the five categories which mentioned above, Thereupon, it does need some manual data cleaning for boosting validation accuracy

Implementation:

We harnessed the most popular approach in AI field which it is Convolutional neural network. With some specific models for distinguish accuracies disparities among them and opt the best models.

Due to the tedious implication of this models from scratch. The transfer model approach will be suitable for that problem. It will be efficient and comprehensible approach. Especially, our problem is computer vision program classification problem².

¹ Unstructured data: Unstructured data has an internal structure, but it's not predefined through data models. It might be human generated, or machine generated in a textual or a non-textual format. Abbreviately, any media or sensor data (audio, images, sensor data, videos Etc.)

² Moreover, AI engineer seldomly construct CNNs models from scratch, unless if the dataset in not universally unfamiliar which it is not in our case

Most importantly, we utilize python programming language with libraries likewise PyTorch, Pandas, Plotly, Matplotlib, NumPy. We utilize COLAB platform due to hardware limitation constraints.

Usually, to make the model more robust we use validation data to make sure our training is going in right path, so we split the training Data into Training data and validation data so we could have 3 types of data sets: Training, Validation and Testing. Then we made A new zip file ready for image classification.

Models:

Resnet:

Revolutionary model which harnesses skip information technique from previous layer to avert vanishing gradient decent³ and provide flexibility to add more layers without drastic decreasing in accuracy of the programs

VGG:

VGG model is an immense upgrade of AlexNet. With some modification in layer, stride and fully connected layer. VGG can be implemented with more than 15 convolutional layers with high and accurate performance

Mobilenet:

In respect to hardware limitation, the depth-wise separable convolutions, which reduce the parameters, could beat this stumbling

ConvNext:

With skip information technique and inverted bottleneck⁴, The recent and the highest accurate program in AI.

The code steps

- *import all require libraries*

```
# License: BSD
# Author: Sasank Chilamkurthy

from __future__ import print_function, division

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import torch.backends.cudnn as cudnn
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
import copy

cudnn.benchmark = True
plt.ion() # interactive mode
```

- *Importing datasets into COLAB*

Passing the google drive link of the data sets through gdown and unzip command

³ Vanishing gradient decent: the gradient learning approaches in back propagation algorithm became vanishingly small. Consequently, the preventing weight from changing there values

⁴ Bottleneck approach: is convolute layers with small kernels and then convolute it again big kernels for comparison the attributes

-

```
[ ] ! gdown --id 1nBNfV9ZATnde3KDPywwu-LTeIXu0kMb1

/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Option '--id' was deprecate
category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1nBNfV9ZATnde3KDPywwu-LTeIXu0kMb1
To: /content/sprints-ai-and-ml-competition-2022 2.zip
100% 264M/264M [00:07<00:00, 36.6MB/s]

[ ] ! unzip "/content/sprints-ai-and-ml-competition-2022 2.zip" -d "/content/Natural Competition"
```

- *Data Augmentation:*

With `transfer.torchvision` method, it shows a noticeable amelioration in our algorithm performance. Despite the amelioration, it vehemently required for transferring model into comprehensible format for the algorithm `totensor` function

`transforms.Normalize` : normalize the tensor image with Z distribution function ⁵

`transforms.RandomHorizontalFlip` : Randomly select images from the data set and flip them horizontally (usually this is made to make the model more robust)

`transforms.ToTensor()` transform the picture into arrays which contains attributes of pixels

```
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize((150,150)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.4302, 0.4575, 0.4543],[0.2362, 0.2345, 0.2433])
    ]),
    'val': transforms.Compose([
        transforms.Resize((150,150)),
        transforms.ToTensor(),
        transforms.Normalize([0.4302, 0.4575, 0.4543], [0.2362, 0.2345, 0.2433])
    ]),
}
```

- *Optimizers and scheduler*

As any Machine learning programs, there is essential need for Optimizer for implicate back propagation and update kernels. Therefore, we utilized two familiar optimizer Adam and SGD in this project.

This is a common step among all transfer models implementation

⁵ $z = (x - \mu) / \sigma$

Adam Function⁶: it more advanced gradient learning algorithm based upon SGD and work efficient with less memory consumption

$$v_t = \beta v_{t-1} + (1 - \beta) * g_t$$

$$w_{t+1} = w_t - \alpha m_t$$

$$m_t = \beta m_{t-1} + (1 - \beta) * g_t^2$$

$$\theta_{t+1} = \theta_t \frac{\eta}{\sqrt{v_t + \epsilon}} m_t$$

```
# Observe that all parameters are being optimized
optimizer_ft = optim.Adam(model_ft.parameters(), lr=0.001)
```

SGD⁷: it like normal gradient decent. However, it updates weights in every iteration rather than in every Epoch. It suitable for enormous datasets with copious of attributes

$$m_t = \beta m_{t-1} + (1 - \beta) * grad(\theta_{t-1})$$

$$\theta_{t+1} := \theta_{t-1} - \alpha m_t$$

```
# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)
```

Scheduler: Decays the learning rate of each parameter group by gamma every step size Epochs. Notice that such decay can happen simultaneously with other changes to the learning rate from outside this scheduler

```
# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

⁶ For more elucidation (Alabdullatef, 2020)

⁷ For more elucidation

```
model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
num_epochs=25)
```

(Srinivasan, 2019)

- *Constructing models*

The common thing between all models is how you can finetuning the transfer model. Thereupon, we firstly acquainting the number of classifier and finetuning into 6 classes -which indicates the five categories where mention above-

For brevity purpose I would prefer to mention python code for one model ⁸

```
model_ft = models.convnext_small(pretrained=True) #
                                                    # if y
last_layer=nn.Linear(768,6)
model_ft.classifier[2]=last_layer

# Alternatively, it can be generalized to nn.Linear
```

As shown above Convnext have 768 clusters (last layer) which will replaced with our 6 clusters

- *Training and Evaluation phase:*

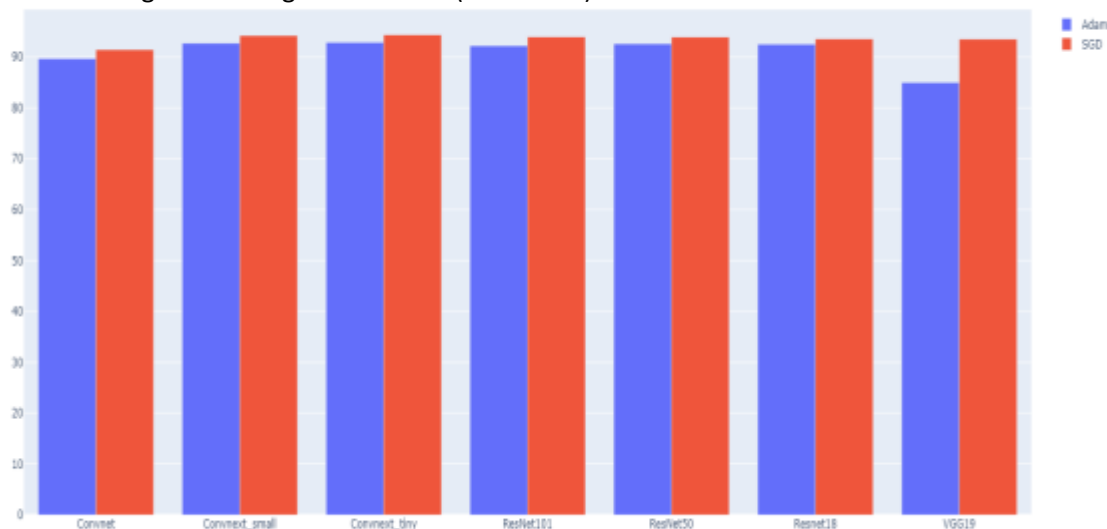
After constructing all the steps above, we passed all this above throughout training function which check if the data

```
model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                        num_epochs=25)
```

Analysis and opting the best model:

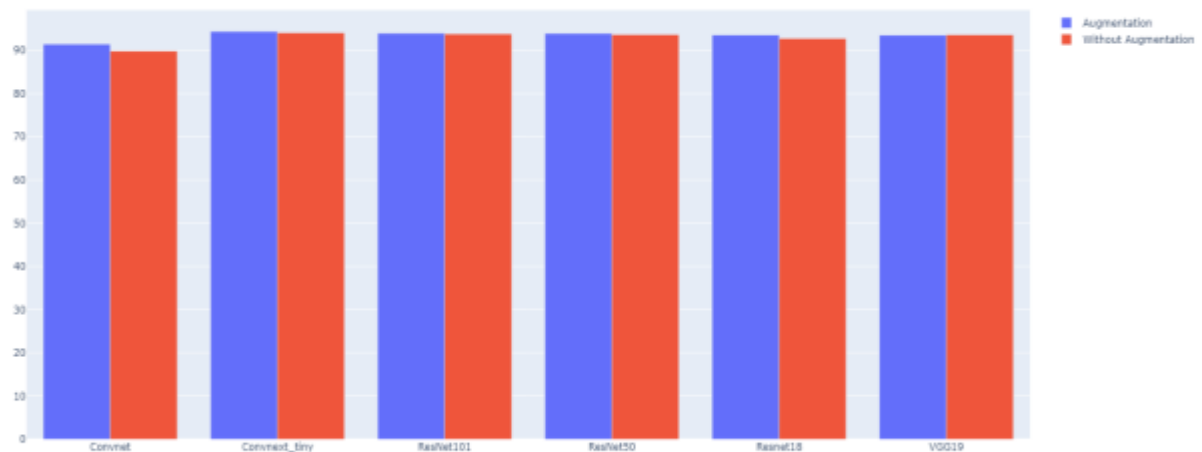
Note: all accuracy number, due to oscillation, cannot be sophisticatedly punctual. However, the fluctuation is not aimed to be vigor, so it prone to be punctual

- The best optimizer: we compared SGD and Adam optimizer in each model. Surprisingly, SGD shows a slight advantage over Adam (trace one)



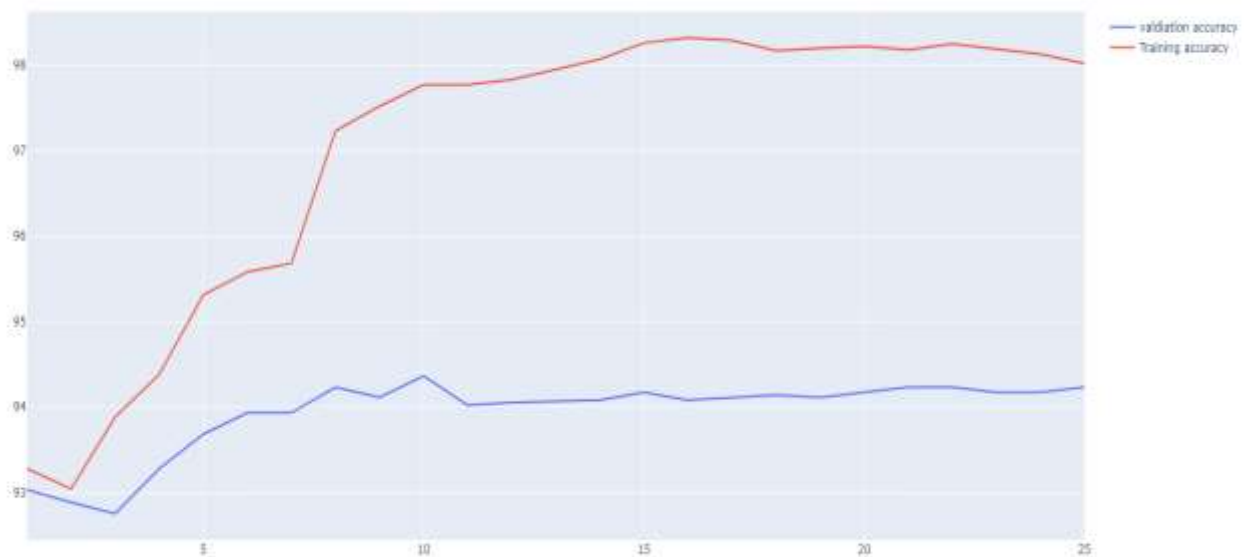
⁸ All finetuning code for each model exists in the official website of PyTorch

- Augmentation, or not: Whilst we did not utilize too much augmentation, we wanted to ascertain about this procedure efficiency



Due to the accuracy nuances between models, it seems unnoticeable by bare looking. However, augmented Convnext tiny model showed great potential with roughly 94.3 accuracy score

- ConvNext is the winner in our competition
Here is multiline graph shows the performance of our transfer model throughout 25 Epochs with SGD Optimizer and Augmentation process



- *Model in-sampling*

However, this is not the end the last step in this training was to combine between all the models and their predictions to make a vote on which image has the most vote on its classification. For instance, if there are four models predicted that a specific image is a Forest, and the other two models predicted that it is Street then it is classified as forest.

The last problem that we faced that the number of models made were even number and there were some images have a draw in their classification and we needed the best and highest accuracy so we decided to add a last model so the total number of models are odd therefore, a final voting could be placed. In addition, we decided to put a restriction, that the model will be added must not be less than 94% validation accuracy so we could insure a fair voting and high accuracy.

Of course this needed to convert the data into a Data Frame so it could be analyzed.





	Image	ConvNextTiny	ConvNextBase	ConvNextSmall	resnet18	resnet50	resnet101	VGG19
0	21689.jpg	0	0	0	0	0	0	0
1	21465.jpg	4	4	4	4	4	4	4
2	21937.jpg	4	4	4	4	4	4	4
3	21857.jpg	5	5	5	5	5	5	5
4	22592.jpg	3	3	3	3	3	3	3
...
3045	22617.jpg	3	3	3	3	3	3	3
3046	23898.jpg	2	2	2	2	2	2	2
3047	21302.jpg	2	2	2	2	2	2	2
3048	23033.jpg	5	5	5	5	5	5	5
3049	22531.jpg	0	0	0	0	0	0	0



3050 rows x 8 columns

Conclusion:

Transfer learning brings a range of benefits to the development process of machine learning models. The main benefits of transfer learning include the saving of resources and improved efficiency when training new models. It can also help with training models when only unlabelled datasets are available, as the bulk of the model will be pre-trained.

The main benefits of transfer learning for machine learning include:

- Removing the need for a large set of labelled training data for every new model.
- Improving the efficiency of machine learning development and deployment for multiple models.
- A more generalised approach to machine problem solving, leveraging different algorithms to solve new challenges.
- Models can be trained within simulations instead of real-world environments.

Saving on training data:

A huge range of data is usually required to accurately train a machine learning algorithm. Labelled training data takes time, effort and expertise to create. Transfer learning cuts down on the training data required for new machine learning models, as most of the model is already previously trained.

In many cases, large sets of labelled data are unavailable to organisations. Transfer learning means models can be trained on available labelled datasets, then applied to similar data that's unlabelled.

Efficiently train multiple models:

Machine learning models designed to complete complex tasks can take a long time to properly train. Transfer learning means organisations don't have to start from scratch each time a similar model is required. The resources and time put into training a machine learning algorithm can be shared across different models. The whole training process is made more efficient by reusing elements of an algorithm and transferring the knowledge already held by a model.

Leverage knowledge to solve new challenges:

Supervised machine learning is one of the most popular types of machine learning today. The approach creates highly accurate algorithms that are trained to complete specific tasks using labeled training data. However once deployed, performance may suffer if the data or environment stray from the training data. Transfer learning means knowledge can be leveraged from existing models instead of starting from scratch each time.

Transfer learning helps developers take a blended approach from different models to fine-tune a solution to a specific problem. The sharing of knowledge between two different models can result in a much more accurate and powerful model. The approach allows for the building models in an iterative way.

Simulated training to prepare for real-world tasks:

Transfer learning is a key element of any machine learning process which includes simulated training. For models that need to be trained in real-world environments and scenarios, digital simulations are a less expensive or time-consuming option. Simulations can be created to mirror real-life environments and actions. Models can be trained to interact with objects in a simulated environment.

Simulated environments are increasingly used for reinforcement machine learning models. In this case, models are being trained to perform tasks in different scenarios, interacting with objects and environments. For example, simulation is already a key step in the development of self-driving systems for cars. Initial training of a model in a real-world environment could prove dangerous and time-consuming. The more generalised parts of the model can be built using simulations before being transferred to a model for real-world training.

References

- [Aishwarya V Srinivasan, Stochastic Gradient Descent — Clearly Explained, Sep 7, 2019, Medium](#)
- [Layan Alabdullatef, Complete Guide to Adam Optimization, Sep 2, 2020, ,Medium](#)
- [Abhijeet Pujara ,Image Classification With MobileNet, Jul 4, 2020, Medium](#)
- [Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, Saining Xie, A ConvNet for the 2020s, 2 Mar 2022, Cornell university](#)
- [Transfer Model For Machine Learning, Jun 29 ,2021, Seldon](#)