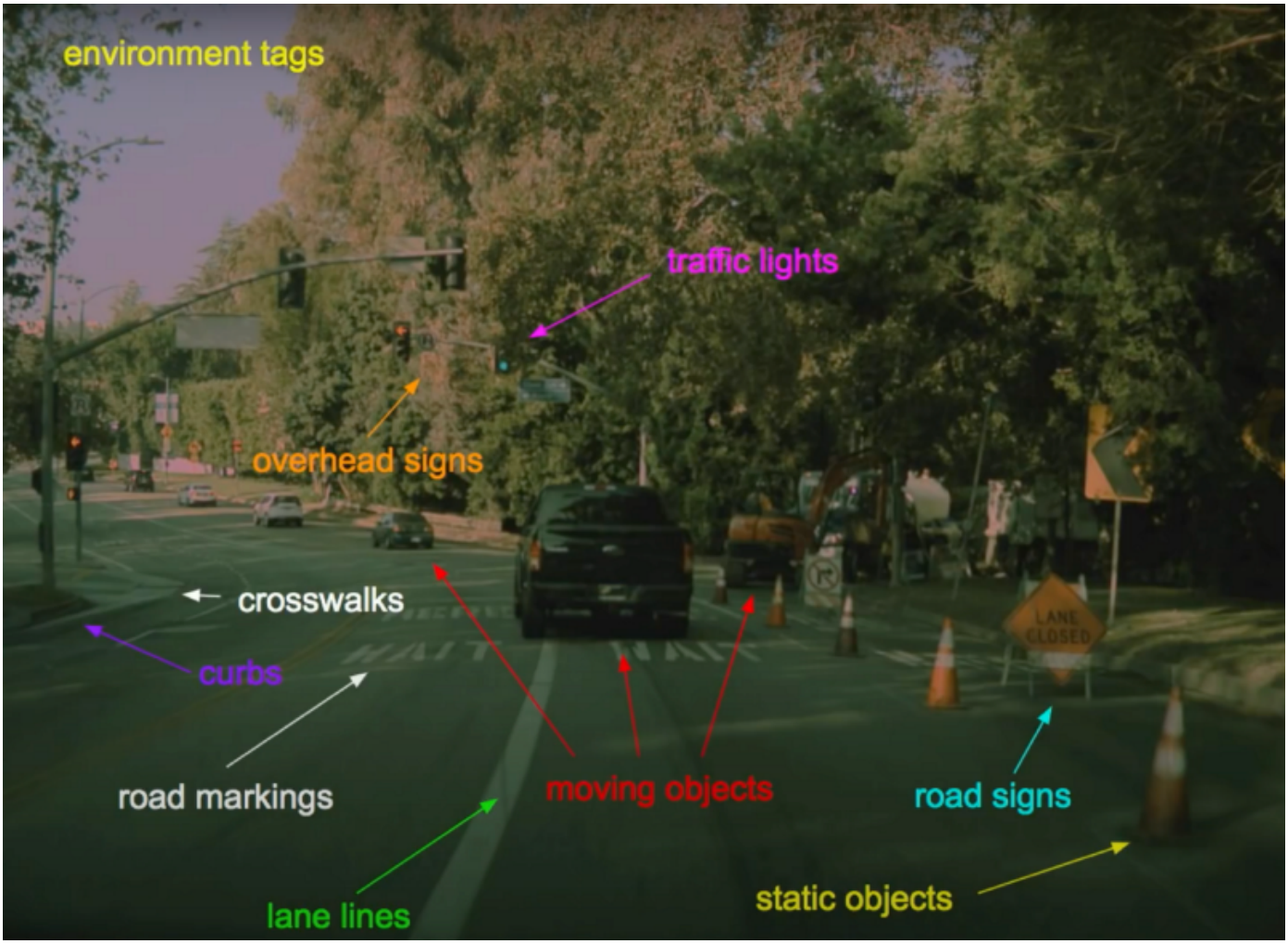
Member-only story

Multi-Task Deep Learning with Pytorch

Alessandro Lamberti · Follow
Published in Artificialis
4 min read · Sep 26, 2022

Listen Share More



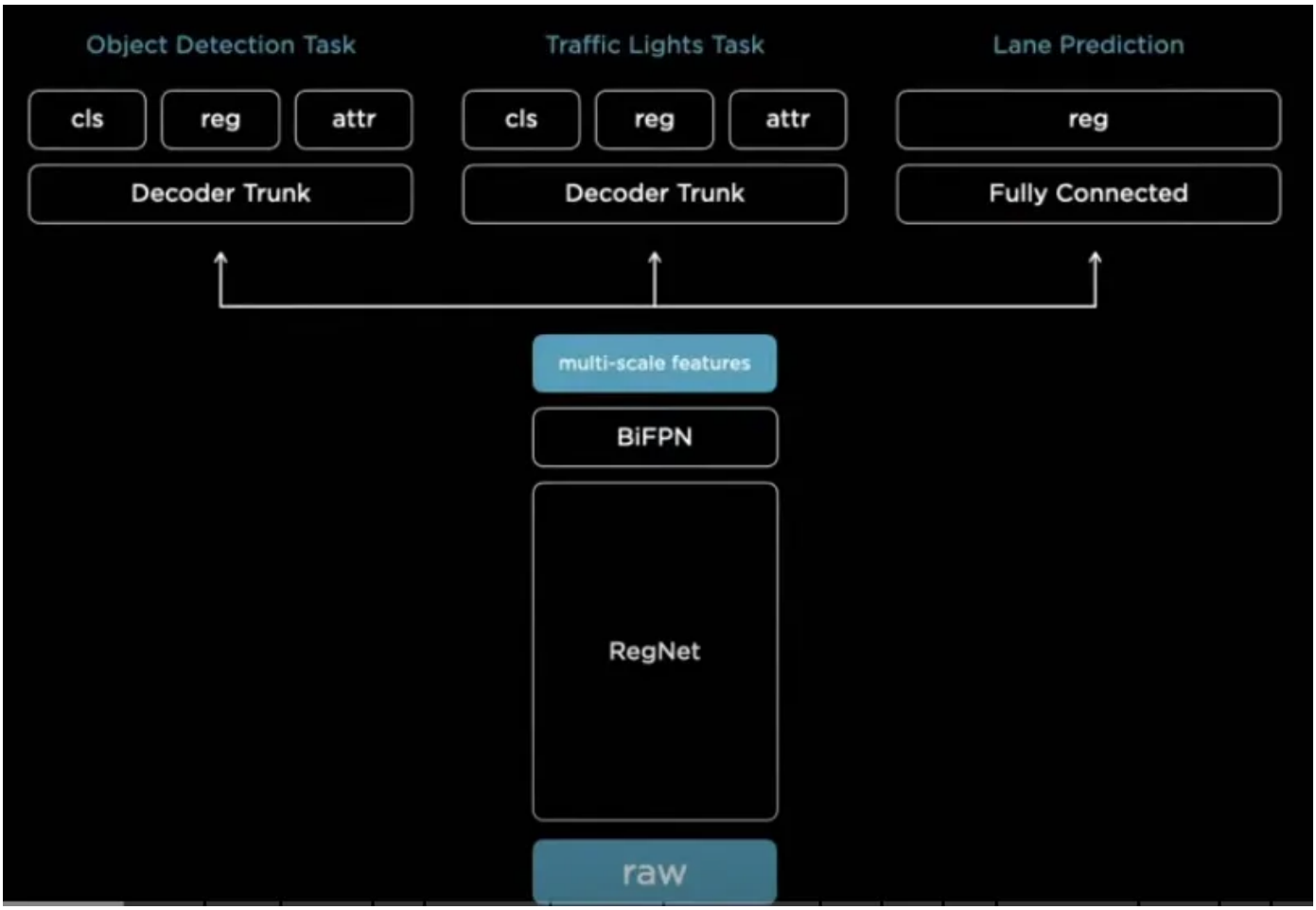
Source

In Machine Learning, we typically aim to optimize for a single metric, for a single task. Multi-task learning (MTL) has led to successes in many applications of machine learning, from natural language processing and speech recognition to computer vision and drug discovery.

The most famous case of MTL is probably Tesla's autopilot.
You already know Tesla's goal is to solve a multitude of tasks like object detection, depth estimation, 3D reconstruction, video analysis, tracking, and more, and you might think they're implementing 10+ Deep Learning models that work together, but that's not the case.

Introduction to HydraNet

Tesla's main idea is fairly simple: one body, several heads. Leveraging a single model to solve multiple tasks.



Source

On a superficial scale, you can see images being extracted by feature extraction models. These images are then combined into a "super image", which are then combined again with previous super images, bringing time into the equation. That's usually done by using 3D CNNs, RNNs, Transformers, etc.

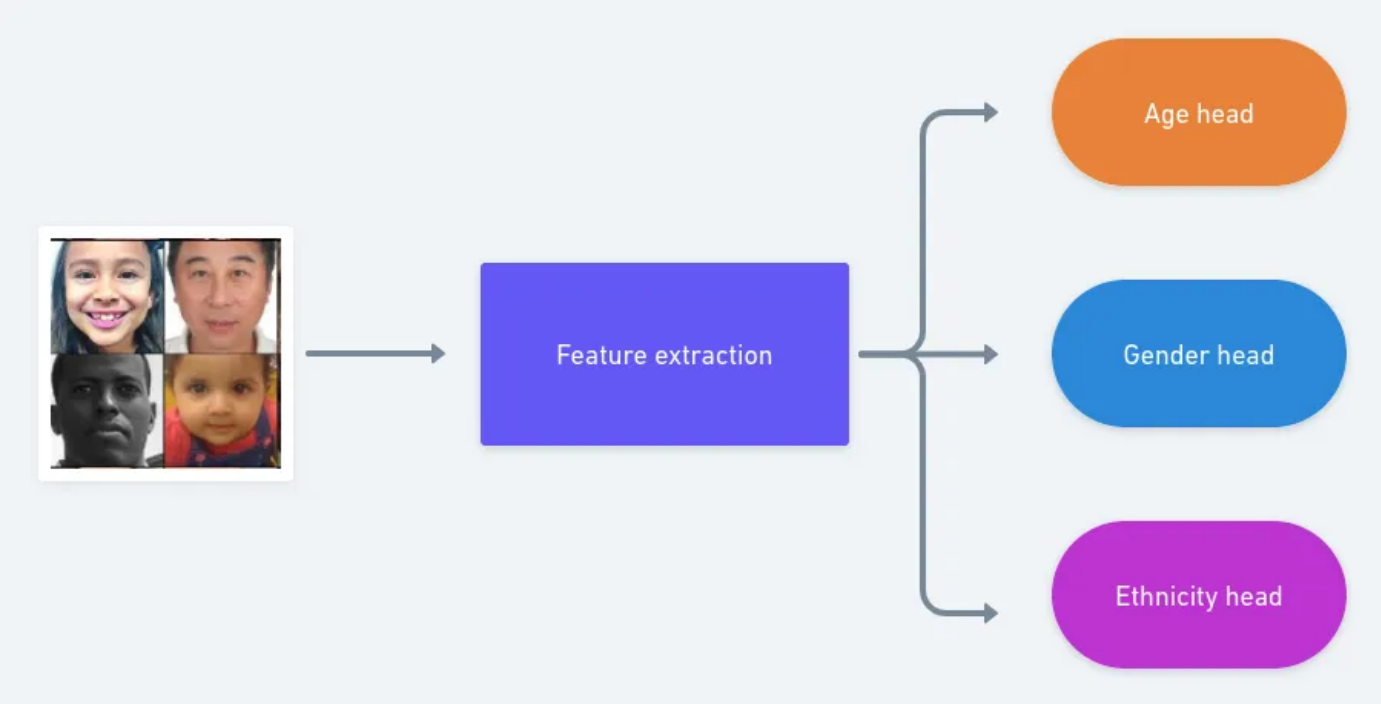
The output is finally split into heads, each one responsible for a specific case, and since they're independent from one another, they can also be fine-tuned individually!

I highly recommend checking out Tesla's keynote.

Multi-task learning project

In this article, we'll see how to implement a simpler HydraNet in Pytorch.
We'll be using the UTK Face dataset, a classification dataset with 3 labels (gender, race, age).

Our HydraNet will have three independent heads, and they'll all be different, because the prediction of age is a regression task, the race prediction is a multi-class classification problem, and the prediction of the gender is a binary classification task.



By the author

Data

We'll start by defining our custom Pytorch Dataset and DataLoader.

Keeping in mind that the name of the images give us the labels, e.g
UTKFace/30_0_3_20170117145159065.jpg.chip.jpg
where

- 30 is the age
- 0 is the gender (0: male, 1: female)
- 3 is the ethnicity (0:White, 1:Black, 2:Asian, 3:Indian, 4:Other)

a custom Dataset might look like this:

```python
class UTKFace(Dataset):
    def __init__(self, image_paths):
        self.transform = transforms.Compose([transforms.Resize((32, 32)), transforms.ToTensor(), t
        self.image_paths = image_paths
        self.images = []
        self.ages = []
        self.genders = []
        self.races = []

        for path in image_paths:
            filename = path[8:].split("_")

            if len(filename)==4:
                self.images.append(path)
                self.ages.append(int(filename[0]))
                self.genders.append(int(filename[1]))
                self.races.append(int(filename[2]))

    def __len__(self):
         return len(self.images)

    def __getitem__(self, index):
            img = Image.open(self.images[index]).convert('RGB')
            img = self.transform(img)

            age = self.ages[index]
            gender = self.genders[index]
            eth = self.races[index]

            sample = {'image':img, 'age': age, 'gender': gender, 'ethnicity':eth}

            return sample
```

dataset_utk.py hosted with ❤ by GitHub view raw

I'm assuming you know your way around Pytorch, but:

- The __init__ method is responsible for the various transformations and the extraction of the labels from the images paths.
- The __get_item__ will do what we defined in the first method: it'll load an image, apply the necessary transformation, get the labels, and it'll return an element of

the dataset.

At this point, we can easily define our data loaders:

```
train_dataloader = DataLoader(UTKFace(train_dataset), shuffle=True, batch_size=BATCH_SIZE)
val_dataloader = DataLoader(UTKFace(valid_dataset), shuffle=False, batch_size=BATCH_SIZE)
```

dataloaders.py hosted with ❤ by GitHub view raw

Model

Now that our data is ready, we can start thinking about the model itself.
We'll be using a pre-trained model as backbone, and we'll build 3 heads, as said before.

At the end, we'll have fc1, fc2 and fc3 for the age's, gender's and ethnicity's head respectively.

```python
class HydraNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = models.resnet18(pretrained=True)
        self.n_features = self.net.fc.in_features
        self.net.fc = nn.Identity()

        self.net.fc1 = nn.Sequential(OrderedDict(
            [('linear', nn.Linear(self.n_features,self.n_features)),
            ('relu1', nn.ReLU()),
            ('final', nn.Linear(self.n_features, 1))]))

        self.net.fc2 = nn.Sequential(OrderedDict(
            [('linear', nn.Linear(self.n_features,self.n_features)),
            ('relu1', nn.ReLU()),
            ('final', nn.Linear(self.n_features, 5))]))

    def forward(self, x):
        age_head = self.net.fc1(self.net(x))
        gender_head = self.net.fc2(self.net(x))
        ethnicity_head = self.net.fc3(self.net(x))
        return age_head, gender_head, ethnicity_head
```

hydranet.py hosted with ❤ by GitHub view raw

As you can notice, the forward method will return every head independently.

Loss

You'll still be trying to minimize a loss, although with a catch: this loss will be the combination of more losses!

The simplest thing one could do is to naively sum the individual losses

$$L = L1 + L2 + L3$$

where:

- L1: the loss associated with the age, like a L1 loss (the mean absolute error), since it's a regression loss.
- L2: the one associated with the ethnicity, like a cross-entropy since it's a multi-class classification loss.
- L3: finally, the one associated with the gender, like binary cross-entropy.

Choosing the approach for the loss function can be tricky, because we're trying to optimize different losses, on different scales.
For example, a loss of 20 years in the age head becomes 400 if we decide to use the mean squared error, compared to the loss of gender which is 1. This will ensure our loss will only be affected by the age, hence we try to compare them on the same scale by using MAE.

Training loop

We'll start by instantiating our HydraNet, the optimizer, and the losses

```
model = HydraNet().to(device=device)

ethnicity_loss = nn.CrossEntropyLoss()
gender_loss = nn.BCELoss()
age_loss = nn.L1Loss()
sig = nn.Sigmoid()

optimizer = torch.optim.SGD(model.parameters(), lr=1e-4, momentum=0.09)
```

trainloop1.py hosted with ❤ by GitHub view raw

And the loop itself

```python
for epoch in range(n_epochs):
    model.train()
    total_training_loss = 0

     for i, data in enumerate(tqdm(train_dataloader)):
         inputs = data["image"].to(device=device)

         age_label = data["age"].to(device=device)
         gender_label = data["gender"].to(device=device)
         eth_label = data["ethnicity"].to(device=device)

         optimizer.zero_grad()
         age_output, gender_output, eth_output = model(inputs)

         loss_1 = ethnicity_loss(eth_output, eth_label)
         loss_2 = gender_loss(sig(gender_output), gender_label.unsqueeze(1).float())
         loss_3 = age_loss(age_output, age_label.unsqueeze(1).float())

         loss = loss_1 + loss_2 + loss_3
         loss.backward()
         optimizer.step()

         total_training_loss += loss
```

trainloop2.py hosted with ❤ by GitHub view raw

That's it! The same process applies for validation as well.

Stay tuned for more advanced things on multi-task learning!

*If you liked the post, consider following me on **Medium***

*You can join **Artificialis** newsletter, here.*

*You can also support my work directly and get unlimited access by becoming a Medium member through my referral link **here**!*

Machine Learning | Tutorial | Artificial Intelligence | Pytorch | Programming

Follow

Written by Alessandro Lamberti

820 Followers · Editor for Artificialis

Machine Learning Engineer | R&D and Intelligence | Sign up: https://alessandro-ml.medium.com/membership

More from Alessandro Lamberti and Artificialis

Alessandro Lamberti in Artificialis

How to extract text from any image with Deep Learning

Learn how to perform OCR with just few lines of python

· 3 min read · Sep 18, 2021

63

Nabeel Khan in Artificialis

Swift and Simple: Calculate Object Distance with ease in just few Lines of Code

So the past few days, I've rummaged through various internet sources looking for ways to calculate object distance using monocular vision...

5 min read · May 26

177

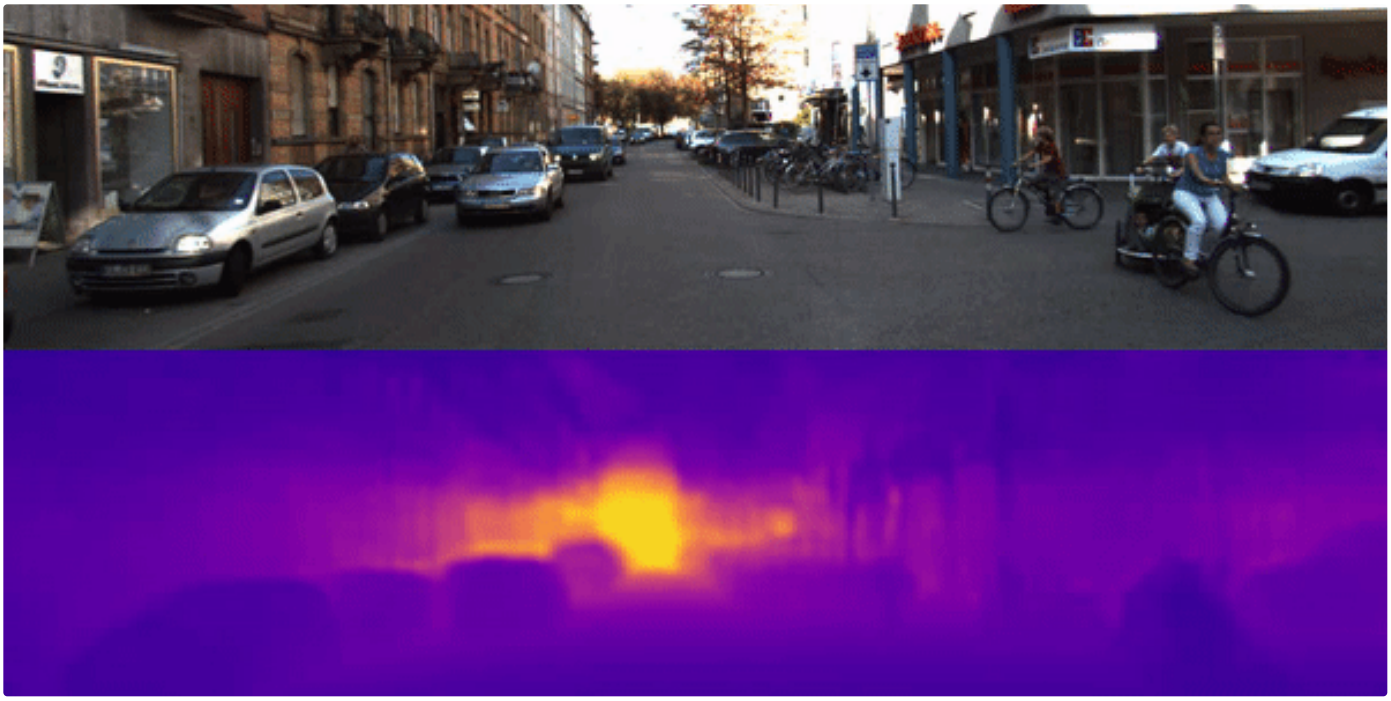
 iva vrtaric in Artificialis

T5 for text summarization in 7 lines of code

Developed by Google researchers, T5 is a large-scale transformer-based language model that has achieved state-of-the-art results on various...

4 min read · Mar 1

 45  

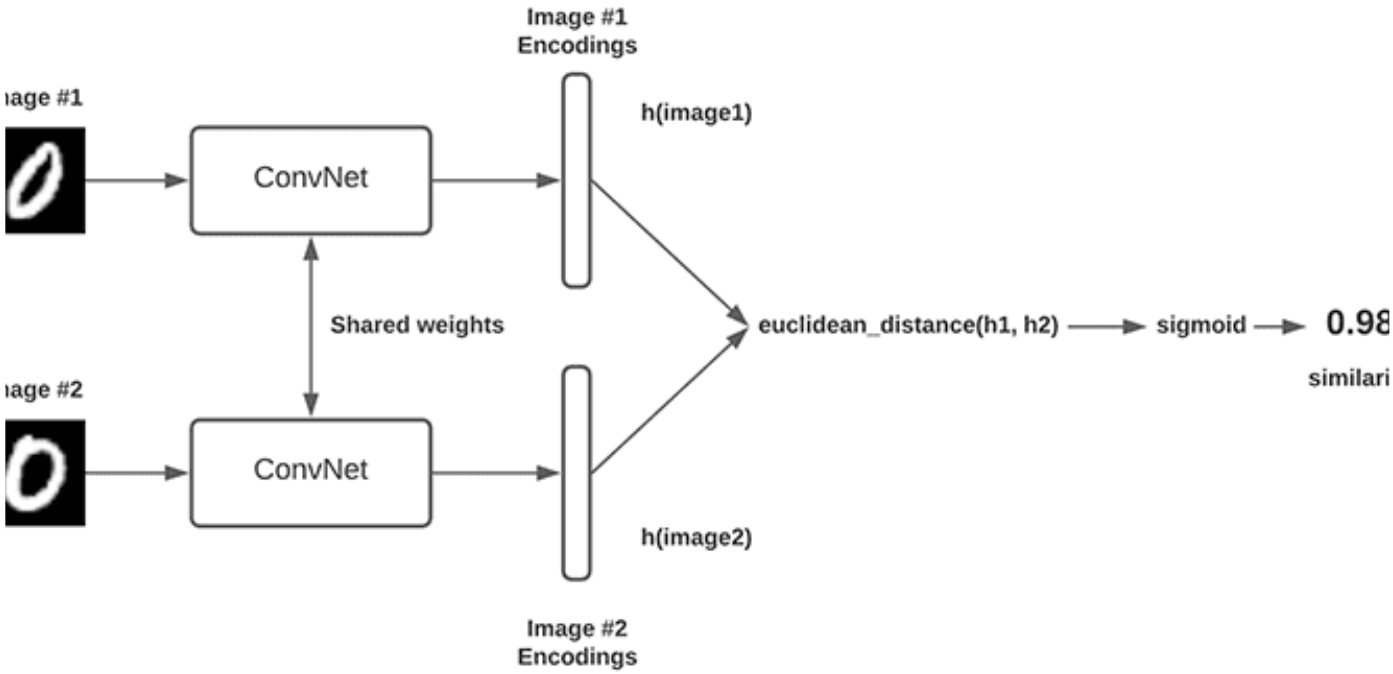


Alessandro Lamberti in Artificialis

Going Deep: An Introduction to Depth Estimation with Fully Convolutional Residual Networks

Have you ever looked at a two-dimensional image and wished you could know the depth of the objects in the scene?

· 10 min read · Feb 27



Prabhat Kumar

Siamese Networks Introduction and Implementation

Artificial Intelligence (AI) has revolutionized numerous industries, and its applications continue to expand. One of the key areas where AI...

5 min read · Jun 25

4 1

Syed Hamza

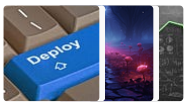
Unlocking the Power of ONNX: A Beginner's Guide with Practical Examples by using 80–20 rule

Discover the game-changing potential of ONNX as we dive into a beginner's guide, packed with practical examples using 80–20 rule.

6 min read · Jul 15


5

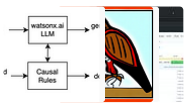
Lists



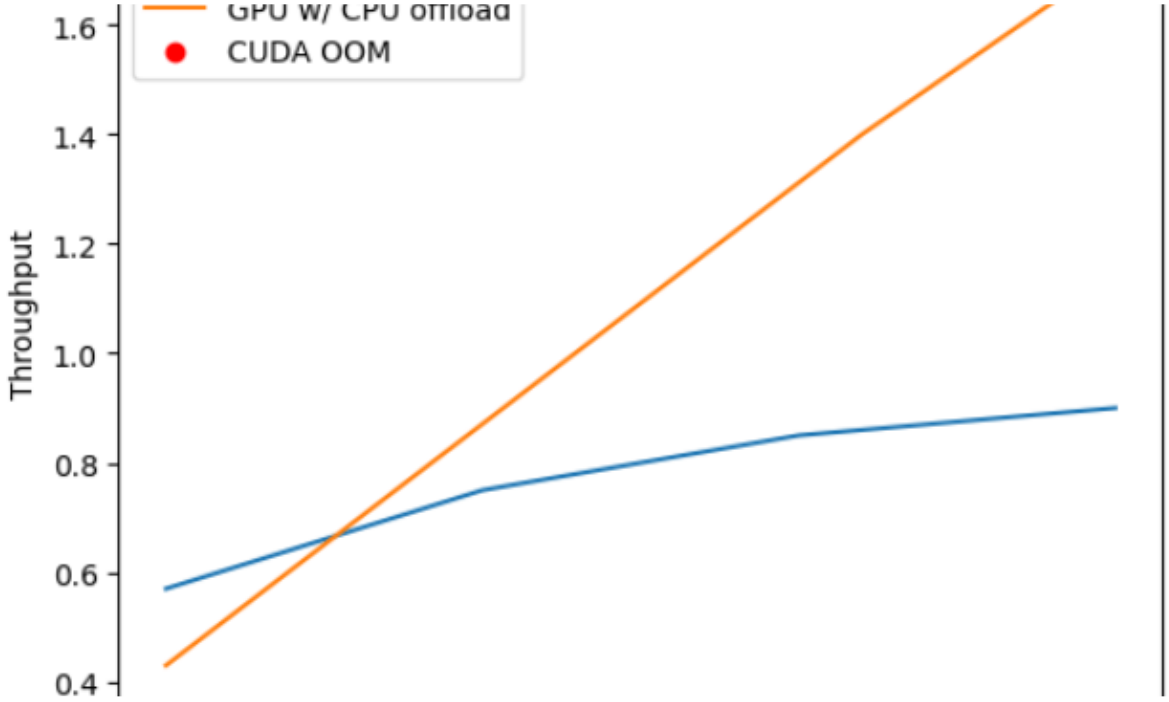
Predictive Modeling w/ Python
20 stories · 300 saves

ChatGPT
21 stories · 120 saves

ChatGPT prompts
24 stories · 280 saves



Natural Language Processing
539 stories · 159 saves



Preemo

Squeeze more out of your GPU for LLM inference—a tutorial on Accelerate & DeepSpeed

by Beite “Jupiter” Zhu

11 min read · Apr 22

9 1

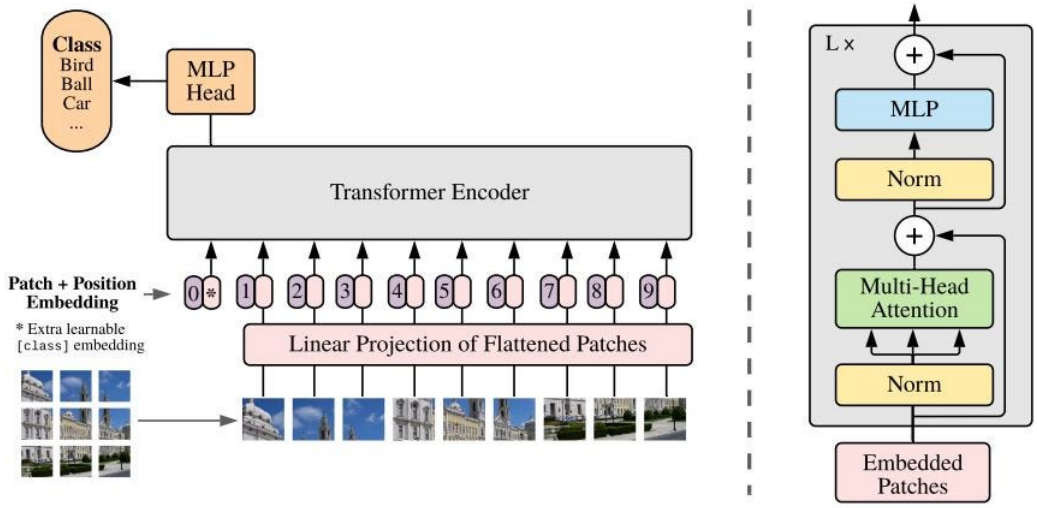


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by

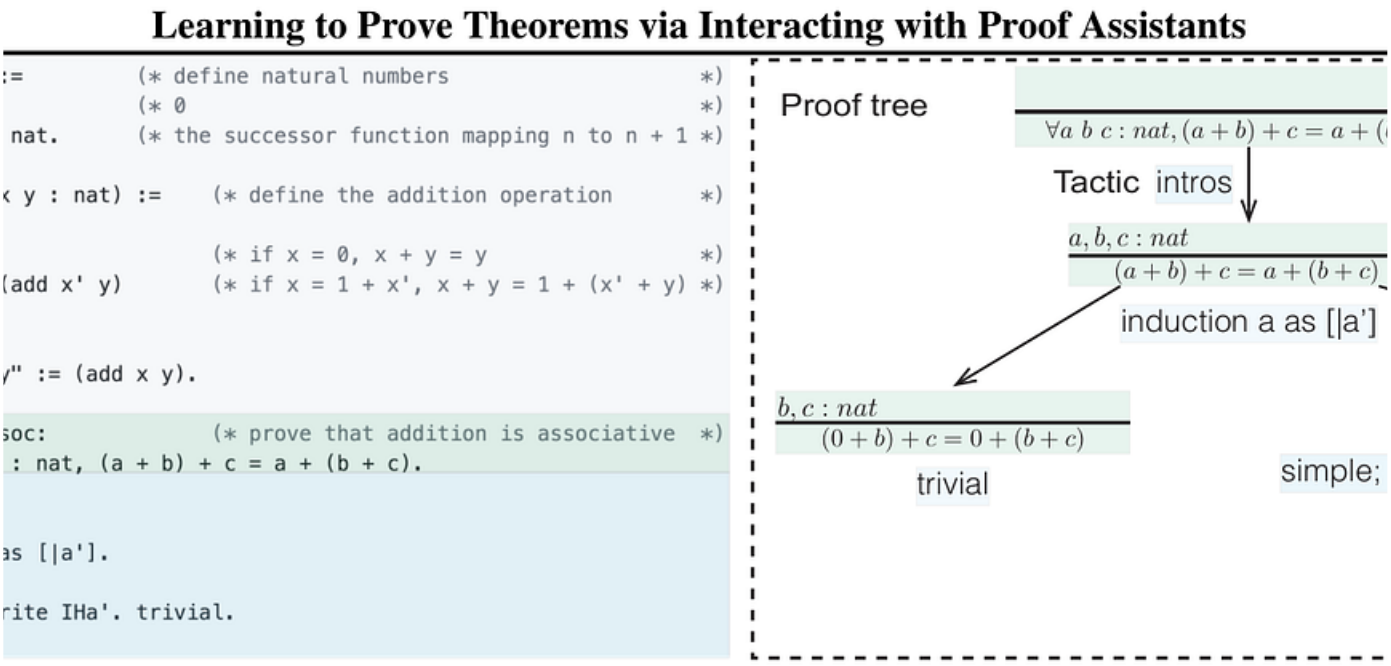
Fahim Rustamy, PhD

Vision Transformers vs. Convolutional Neural Networks

This blog post is inspired by the paper titled AN IMAGE IS WORTH 16×16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE from google's...

7 min read · Jun 4

356 6



Daniel Jenson in Stanford CS224W GraphML Tutorials

Automated Theorem Proving with Graph Neural Networks

By Dan Jenson, Julian Cooper, and Daniel Huang as part of the CS 224W course project at Stanford University.

10 min read · May 15

Utsav Shrestha

JAX vs. PyTorch: A Comprehensive Comparison for Deep Learning

Deep learning has become a fundamental part of modern machine learning, and choosing the right library is crucial for success. JAX and...

3 min read · Jun 5

 19