Transfer Learning

An AI Commons initiative

Contents

1. The Current State of Deep Learning training
2. Initial Approach
3. Transfer Learning
4. Fine Tuning
5. Transfer Learning in CNNs
6. Transfer Learning: How much to train?
7. References

The current state of Deep Learning training

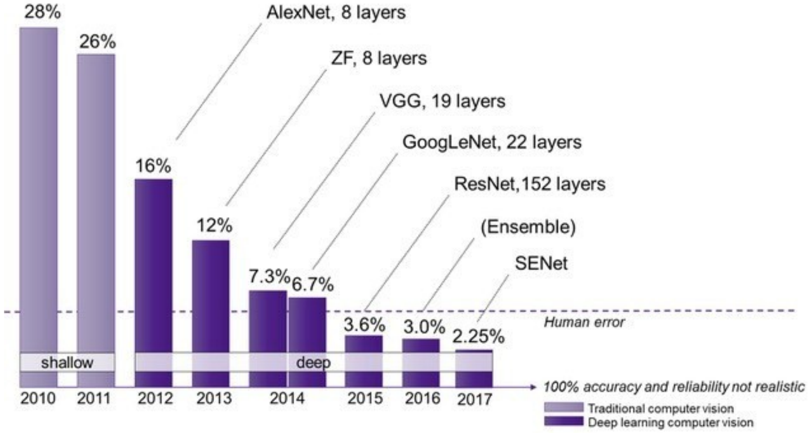
The most recent breakthroughs in DL have also come with the predictable problem of increased computational power and time needed in training.

Is there a way to train complex architectures for our applications with easy to access hardware in a reasonable time?

Initial Approach

In the internet there is already a multitude of neural nets trained in huge datasets like ImageNet, COCO, Pascal and many more with great performances.

This networks could have taken weeks of training in very powerful GPUs.



https://semiengineering.com/new-vision-technologies-for-real-world-applications/

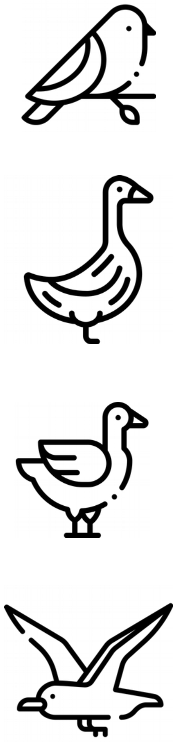
Initial Approach

Instead of repeating the training process, let's use these trained weights as the starting point for our training. Effectively ‘transferring’ what they learned to our model.

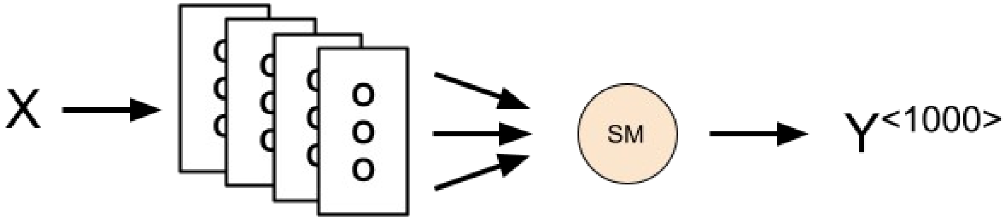
This could also help in cases where we don't have many samples in our dataset.

Transfer Learning

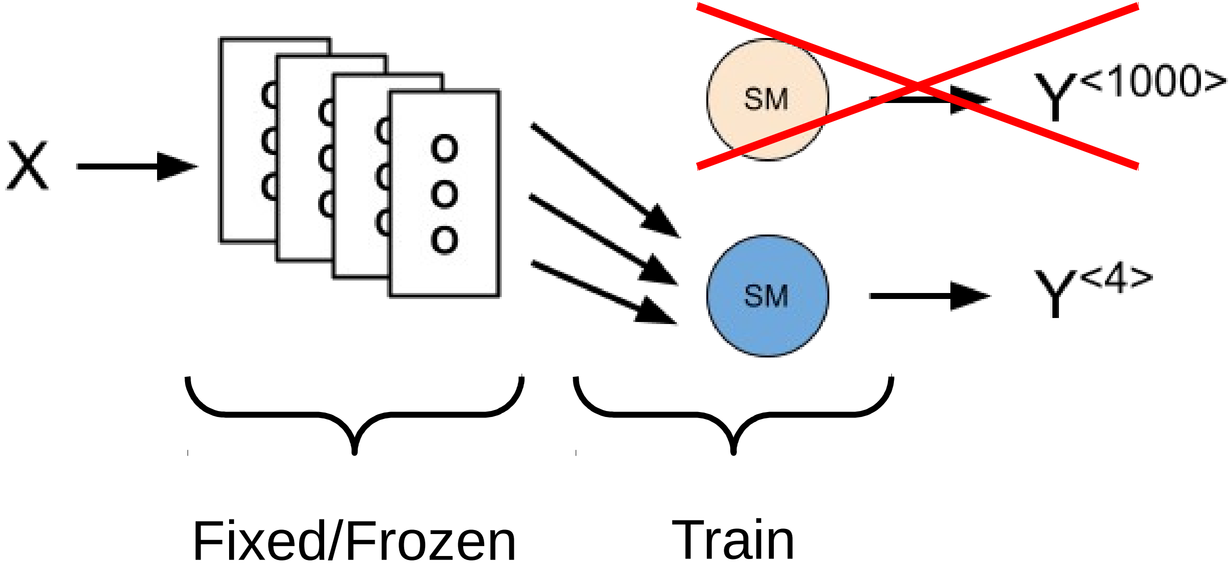
Let's suppose we are tasked with the objective of training a classifier for some common species of birds, but the amount of data that we have is not enough to train a CNN from scratch.



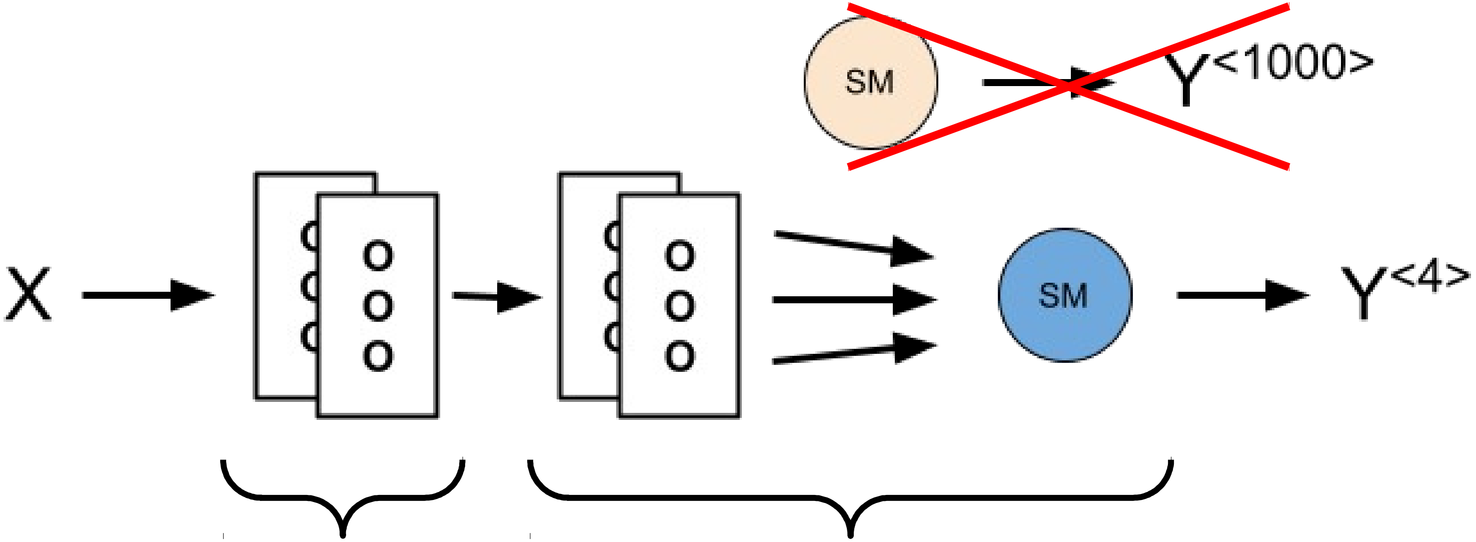
Instead, lets download a model and its weights from the internet trained over a 1000 classes, like the following net:



Transfer Learning



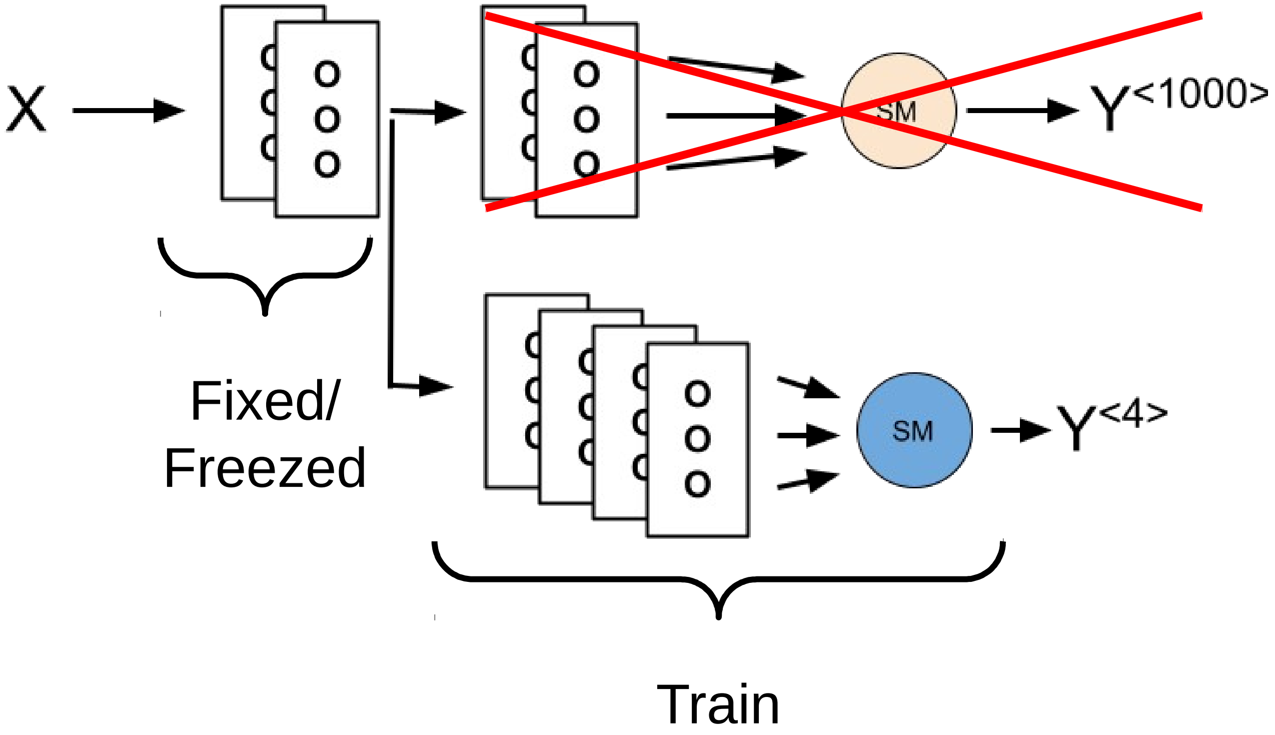
Transfer Learning



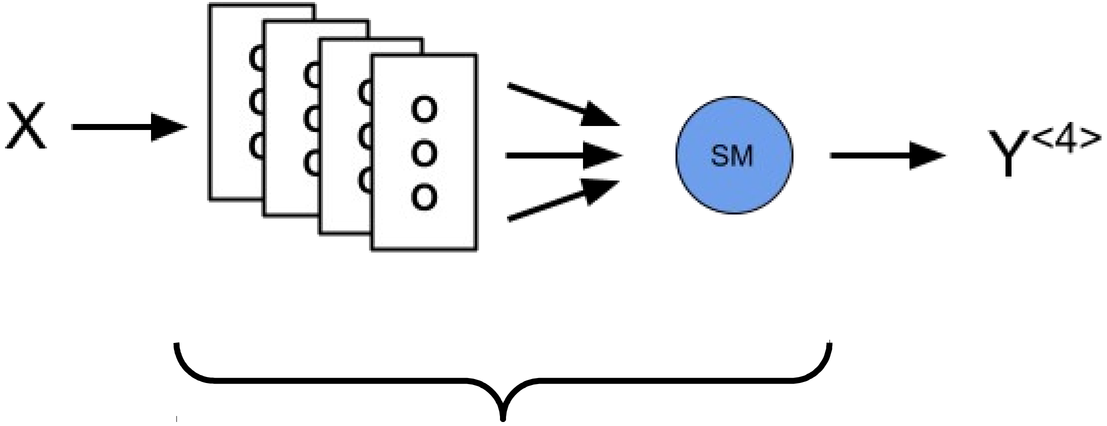
Fixed/
Frozen

Train

Transfer Learning



Fine Tuning

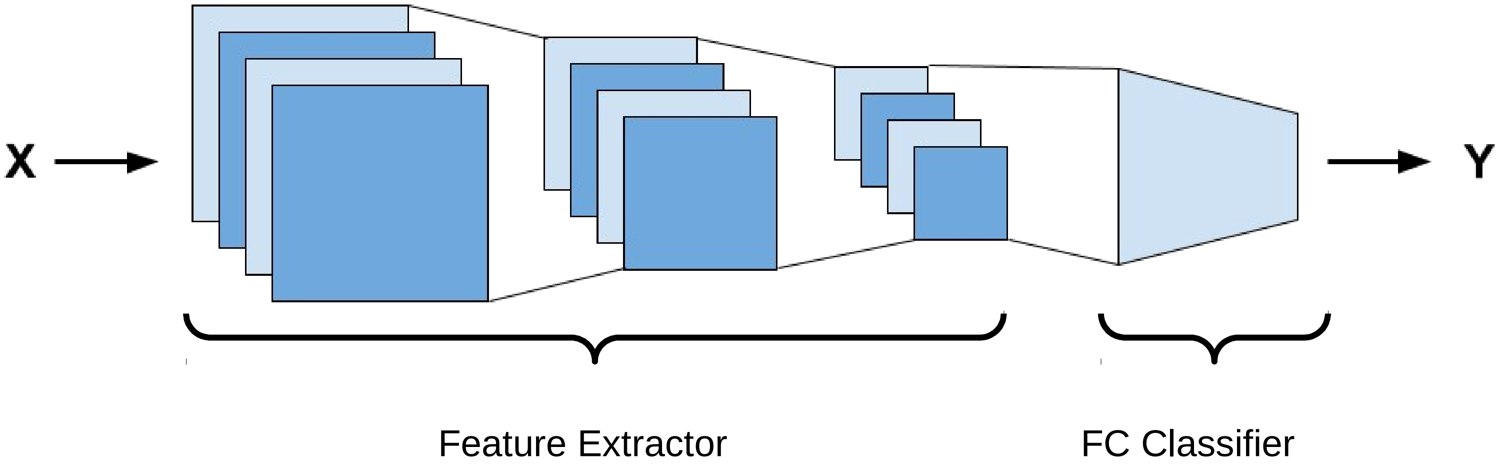


Train

In Fine Tuning it is important to remember to use a small learning rate.

Transfer Learning in CNNs

Classic CNNs follow a general structure that is important to keep in mind before using transfer learning.



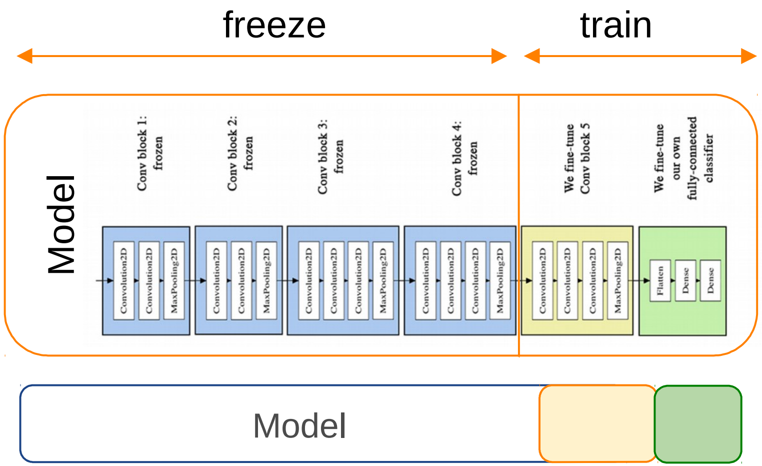
Transfer Learning in CNNs

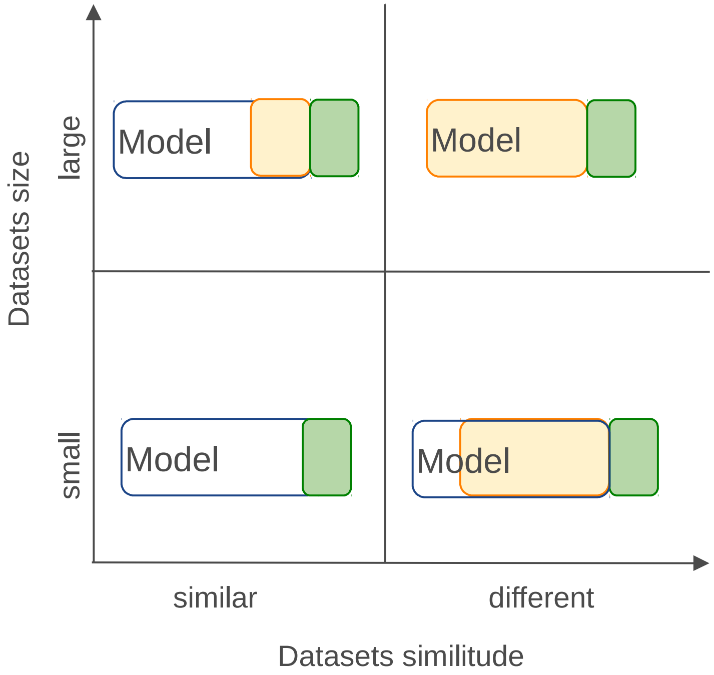
The Fully Connected (FC) part of a CNN is often discarded when using Transfer Learning, while keeping the convolutional layers.

This is because the convolutional layers extract the features from the input.

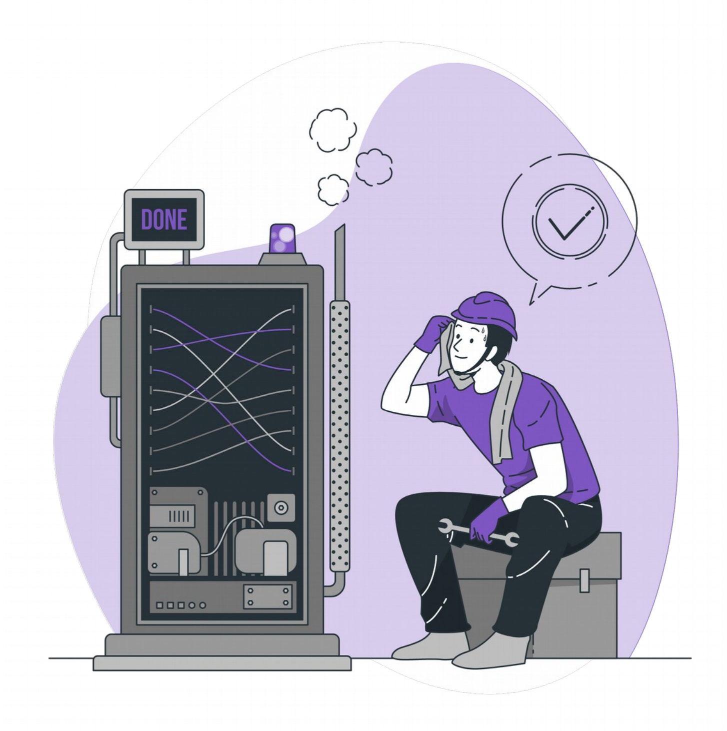
If extra layers are added, those have to be after the frozen layers, so the learned information is not lost.

Transfer Learning: How much to train?





That's it!



Extra Utility: Dataloaders

Up to now, we have loaded the whole dataset in the program for training. But in cases when the data weights multiple gigabytes, this might be impossible with a normal computer.

Instead, Dataloaders offer a way to load only the necessary data from a hard drive for each batch. This is slower than just loading the whole dataset, but allows to work with more data.

Some ML libraries come with built-in Dataloaders, and others offer tools to build your own.

Building Dataloaders: Pytorch

```
class Dataset(torch.utils.data.Dataset):
  'Characterizes a dataset for PyTorch'
  def __init(self, list_IDs, labels):
        'Initialization'
        self.labels = labels
        self.list_IDs = list_IDs

  def __len__(self):
        'Denotes the total number of samples'
        return len(self.list_IDs)

  def __getitem(self, index):
        'Generates one sample of data'
        # Select sample
        ID = self.list_IDs[index]

        # Load data and get label
        X = torch.load('data/' + ID + '.pt')
        y = self.labels[ID]

        return X, y
```

Building Dataloaders: Pytorch

```
training_set = Dataset(partition['train'], labels)
training_generator = torch.utils.data.DataLoader(training_set, **params)

validation_set = Dataset(partition['validation'], labels)
validation_generator = torch.utils.data.DataLoader(validation_set, **params)

# Loop over epochs
for epoch in range(max_epochs):
    # Training
    for local_batch, local_labels in training_generator:
        [...]

    # Validation
    with torch.set_grad_enabled(False):
        for local_batch, local_labels in validation_generator:
              [...]
```

Building Dataloaders: Keras

```
tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    zca_epsilon=1e-06,
    rotation_range=0,
    width_shift_range=0.0,
    height_shift_range=0.0,
    brightness_range=None,
    shear_range=0.0,
    zoom_range=0.0,
    channel_shift_range=0.0,
    fill_mode="nearest",
    cval=0.0,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=None,
    preprocessing_function=None,
    data_format=None,
    validation_split=0.0,
    dtype=None,
)
```

Building Dataloaders: Keras

```
datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True)

datagen.fit(x_train)

model.fit(datagen.flow(x_train, y_train, batch_size=32),
          steps_per_epoch=len(x_train) / 32, epochs=epochs)
```

References

https://keras.io/guides/transfer_learning/

https://www.coursera.org/lecture/convolutional-neural-networks/transfer-learning-4THzO

https://cs231n.github.io/transfer-learning/

https://machinelearningmastery.com/transfer-learning-for-deep-learning/