1. Transfer learning

* A very important topic in modern deep learning

A diagram of a training curve

Description automatically generated

1. Basic concepts

* Recall: features are hierarchical (CNN)
* Notice: all CNNs learn simple lines and strokes

1. Transfer learning intuition

* Main idea: the features I found from one task may be useful for another task
* Transfer learning took off in the field of computer vision
* ImageNet – large-scale image dataset (millions of images, 1k categories)
* Because the dataset is so diverse, weights trained on this dataset can be applied to a large number of vision tasks
* Cats vs. dogs
* Cars vs. trucks
* Even microscope images / images never seen before

1. Training on ImageNet

* Not feasible for us to train on ImageNet ourselves
* Old days: training used to take days, weeks, even months
* Now we can do it in minutes, using multi-GPU clusters, etc.
* The major CNNs which have won past ImageNet contests have publicly released pre-trained weights
* No need to do hyperparams tuning, etc.
* They are already included in Tensorflow/Keras

1. 2-part CNN

A diagram of a transformer

Description automatically generated

1. Transfer learning in practice

* Chop off the old ‘head’, add a new head!

A diagram of a graph

Description automatically generated

* Can be logistic regression or ANN
* E.g., ‘cars vs. trucks’ may use a logistic regression with sigmoid
* Freeze the ‘body’
* Train only the head (much faster!)

A screenshot of a graph

Description automatically generated

1. Advantages of transfer learning

* Don’t need a lot of data to build a state-of-the-art model
* With transfer learning, this work has been done for us – the earlier features were already trained on lots of data
* Small dataset + a lot less weights helps us train faster

A diagram of a data

Description automatically generated

1. Some pre-trained models

* VGG
* Named after the research group creating it – Visual Geometry Group
* Not that different from CNNs we already know, just bigger
* Options: VGG16, VGG19

A screenshot of a computer

Description automatically generated

* ResNet
* A CNN with branches (one branch is the identity function, so the other learns the residual)
* Variations: ResNet50, ResNet101, ResNet152, ResNet\_v2, ResNeXt

A screenshot of a computer

Description automatically generated

* Inception
* Multiple convolutions in parallel branches
* Instead of trying to choose different filter sizes (1x1, 3x3, 5x5, etc.), just try them all!

A diagram of a tree

Description automatically generated

* MobileNet
* Lightweight: makes a tradeoff between speed and accuracy
* Meant for less powerful machines (mobile, embedded)

A diagram of a cube

Description automatically generated

* Preprocessing:
* Since we are using pre-built CNNs, our data must be formatted just like the original
* We usually work with RGB pixel values in [0, 1] or [-1, 1]
* VGG uses BGR with pixels centered but NOT scaled
* Import the preprocess\_input function from the same module as your pretrained network

A screen shot of a computer

Description automatically generated

1. Large datasets and data generators

* Large image datasets
* When you are first learning about deep learning, it is convenient to have MNIST, CIFAR-10, SVHN, etc. as a CSV/numpy array
* In the real world, images are not CSVs, images are images (i.e., image files .jpeg, .png, etc.)
* Real images are much larger than MNIST and CIFAR
* VGG and ResNet are trained on ImageNet images resized to 224x224
* How much space does it take to store 1 million images of size 224x224?
* 1 million images x (224x224x3) bytes / image
* ~ 150 billion bytes ~ 140GB
* Would not fit in RAM on a standard machine
* What’s the solution?
* Approach the problem like an engineer, not that of a simple API user
* Recognize the difference between disk and memory
* Disk: slow >< memory: fast
* Model reads data from memory; images files live on disk
* We do batch gradient descent (only the current batch needs to exist in memory)
* Assume we have 2 arrays: list of filenames, list of labels
* Suppose batch\_size = 32

A close-up of a math problem

Description automatically generated

* How we actually do it:
* Main ingredients:
  + gen = ImageDataGenerator()
    - Automatically generates data in batches
    - Data augmentation (shifting, rotation, flipping, etc.)
    - Preprocessing (via preprocess\_input)
  + generator = gen.flow\_from\_directory()
    - specify target image size, batch size
  + model.fit\_generator(generator)
    - used in place of model.fit(x, y)
* Folder structure

A screenshot of a computer

Description automatically generated

1. 2 approaches to transfer learning

* Suppose the body has 100s of layers, but you have only 1 head layer
* Still takes time to compute the output prediction
* 2-part CNN:
* Imagine the computation in 2 parts:
  + Part 1: z = f(x) # pre-trained CNN – slow
  + Part 2: y\_hat = softmax(Wx + b) # logistic regression – fast

A black text on a white background

Description automatically generated

* Does the calculation of z ever change? No
* All data is the same:
* Z = f(x) # use CNN to precompute all z’s at once
* Turn (z, y) into a tabular dataset
* Fit (z, y) to logistic regression, never have to look at x again!
* Problem:
* How can we use data augmentation
* If x is always slightly different, then so is z!
* The 2 approaches:
* Approach 1: use data augmentation with ImageDataGenerator
  + Entire CNN computation must be inside loop
  + Slow (must pass through entire CNN)
  + Possibly better for generalization
* Approach 2: precompute z without data augmentation
  + Only need to train a logistic regression on (z, y)
  + Fast (only need to pass through 1 dense layer)
  + Possibly worse for generalization