

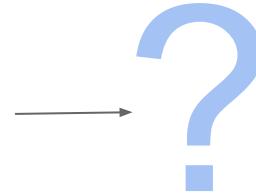
PREDICTING MY SON'S MOOD WITH DEEP LEARNING

Dan Howarth

NEURAL NETWORKS ARE HARD TO TRAIN...



New Parents ~
no labelled data



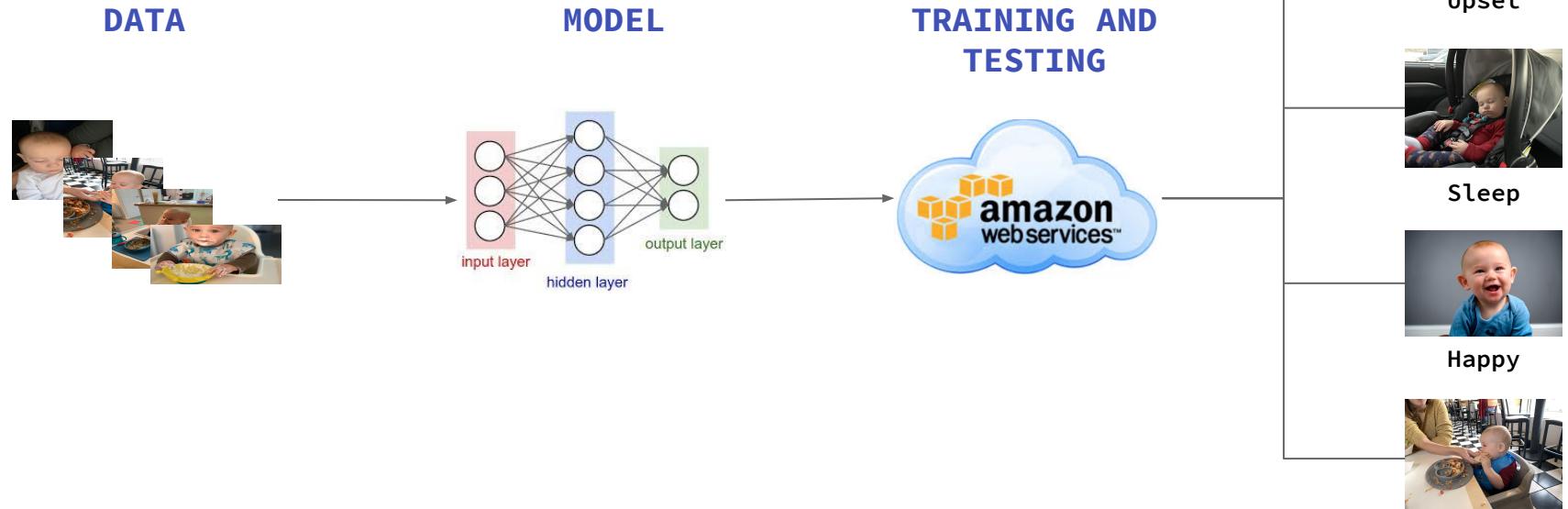
Neural Network a.k.a.
Harper



Grandparents ~
data old and corrupted

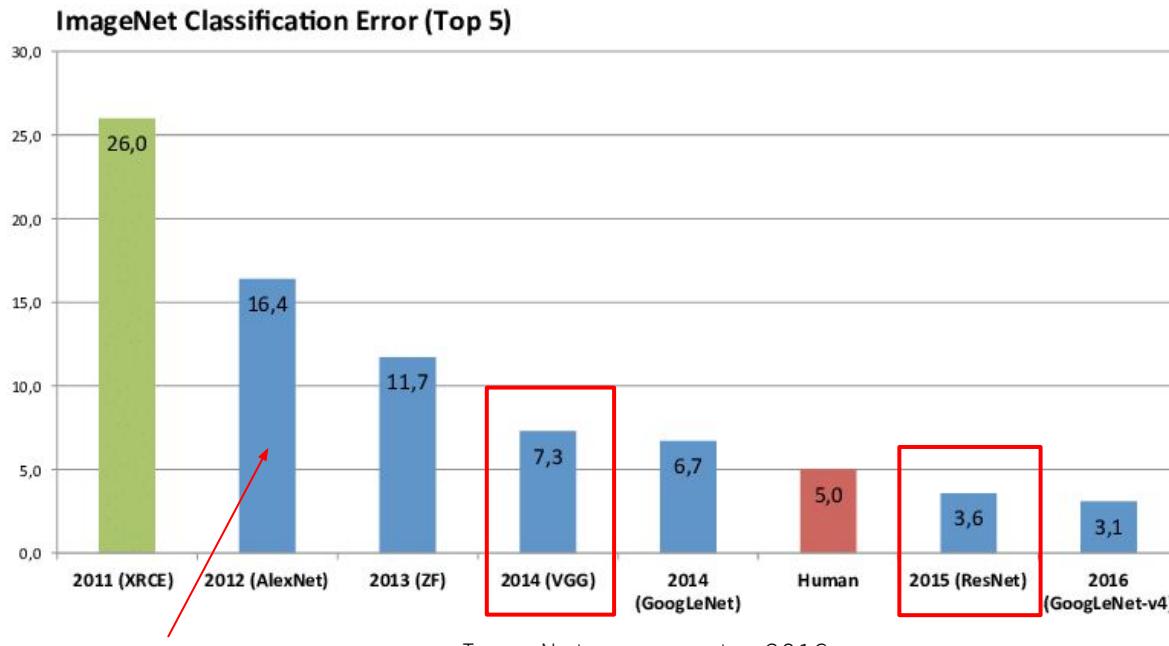
...HOPEFULLY ARTIFICIAL ONES ARE A BIT EASIER

MY AIM WAS TO BUILD A PREDICTIVE MODEL...



...AND LEARN MORE ABOUT COMPUTER VISION IN THE PROCESS

WHAT IS COMPUTER VISION?



Computer Vision includes
image recognition, object
detection

ImageNet is a database of
over 15 million images – and
an annual competition that
drives State of the Art in
Computer Vision

DATA

WHAT MAKES A GOOD DATASET?

(Good) Dataset Challenges...a face that changes over time...



Representative



Varied

Balanced classes

Different train, val and test sets

...and one class (Sleep) represented in different ways

HOW MUCH DATA IS ENOUGH?



Training models from scratch requires extremely large datasets

Data Augmentation and Transfer Learning partly reduces this requirement

Augmentation gives the model a different version of the image at training

DATA AUGMENTATION REDUCES DATASET REQUIREMENTS

```
from torch.utils.data import Dataset

class HarperNetDataset(Dataset):
    ''' class to process the HarperNet dataset'''

    def __init__(self, dataset, transform = None):

        self.dataset = dataset
        self.transform = transform

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

    def __getitem__(self, index):

        X, y = self.dataset[index][0], self.dataset[index][2]

        if self.transform:
            X = self.transform(X)

        return X, y

train_dataset = HarperNetDataset(train_data,
                                transform=train_transform)
```

HARPERNET: INSTANTIATING A DATASET

Create our own dataset class that inherits from PyTorch's Dataset class

Override the `__len__()` and `__getitem__()` function

Enable the dataset to be transformed

Return dataset as an x,y pair or dictionary

Returns the data and corresponding variables

HARPERNET: TRANSFORMING AND PROCESSING DATASET

```
# train transforms
train_transform = transforms.Compose([transforms.ToPILImage(),
                                    transforms.RandomResizedCrop(224),
                                    transforms.RandomRotation(degrees=15),
                                    transforms.ColorJitter(),
                                    transforms.RandomHorizontalFlip(),
                                    transforms.ToTensor(),
                                    transforms.Normalize([0.485, 0.456, 0.406],
                                                       [0.229, 0.224, 0.225]))]

# validation and test transforms
valtest_transform = transforms.Compose([transforms.ToPILImage(),
                                       transforms.Resize(256),
                                       transforms.CenterCrop(224),
                                       transforms.ToTensor(),
                                       transforms.Normalize([0.485, 0.456, 0.406],
                                                          [0.229, 0.224, 0.225]))]
```

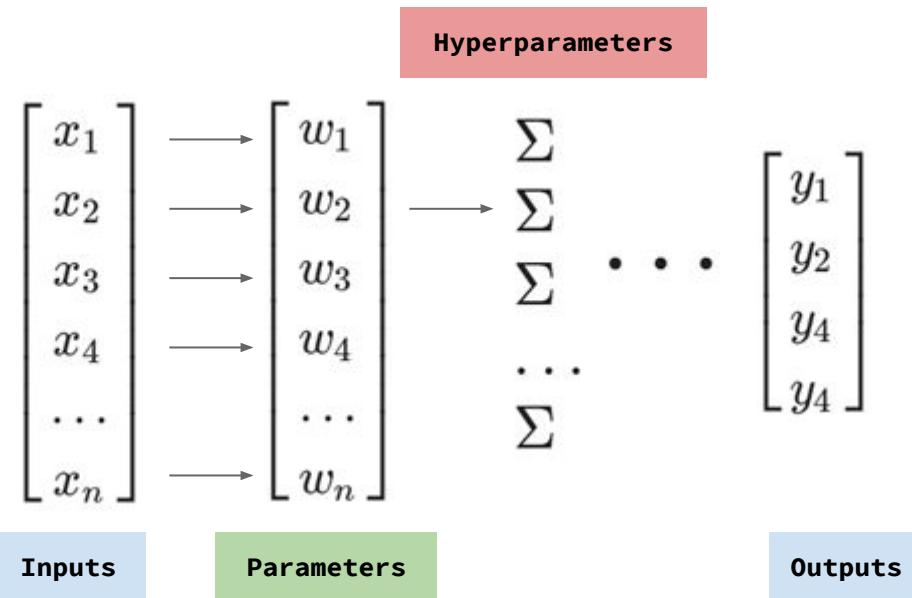
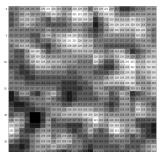
Train transformations augment the images and will be applied at training time

Validation and test transformations apply ImageNet standards and convert data to tensor data type

MODEL



HOW DOES DEEP LEARNING WORK?



Deep learning models the relationships between inputs and outputs

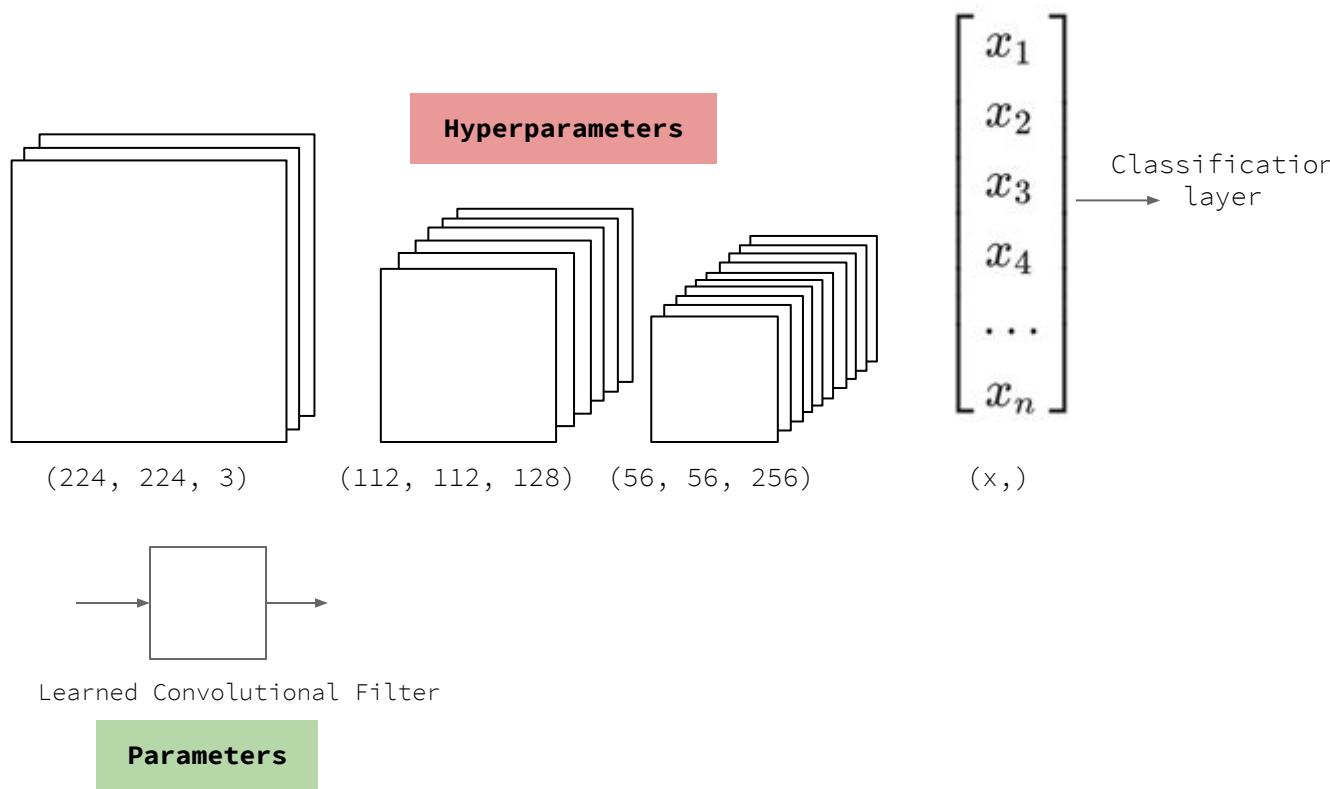
Hyperparameters include the number and size of layers and the activation functions

Parameters we train are the weights in between layers

Values are passed through the layers, multiplied by weights and then an activation function

DEEP LEARNING MODELS ARE AN ART AS MUCH AS A SCIENCE

WHY ARE CNNS USED FOR COMPUTER VISION?



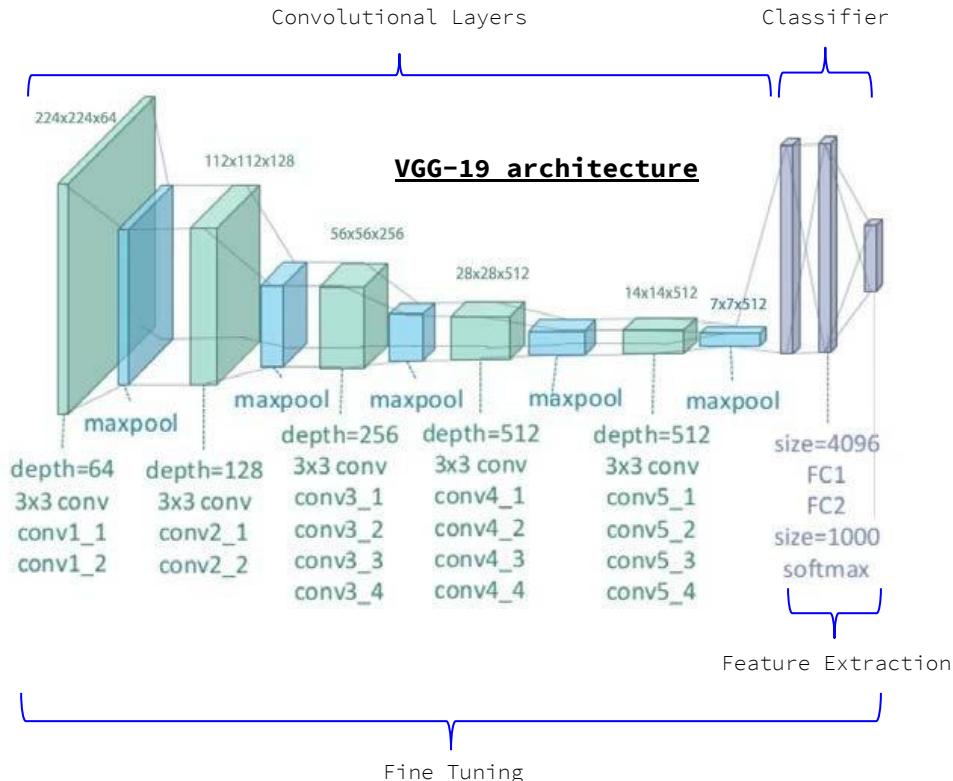
CNNs produce a better representation of the data before we go to the classifier stage

The image tensor is passed into the model in its current shape

Convolutional filters at each layer extract information about the images

Fully Connected Layers are now the classifier at the end of the model

WHAT IS TRANSFER LEARNING?



Transfer Learning allows us to use state of the art models

The transfer the model architecture and the learnt weights

Models can be used as a feature extractor or fine tuning, and be modified

```
from torch import nn
from torchvision import models
import torch.nn.functional as F

class ResNetCNN(nn.Module):

    def __init__(self, class_size):
        super(ResNetCNN, self).__init__()

        resnet = models.resnet101(pretrained=True)

        for param in resnet.parameters():
            param.requires_grad_(False)

        modules = list(resnet.children())[:-1]

        self.resnet = nn.Sequential(*modules)
        self.category = nn.Linear(resnet.fc.in_features, class_size)

    def forward(self, images):
        features = self.resnet(images)

        features = features.view(features.size(0), -1)

        features = F.softmax(self.category(features))

        return features
```

HARPERNET: TRANSFER LEARNING IMPLEMENTATION

One line of code downloads a pretrained ImageNet model

We specify that the model parameters shouldn't be retrained – set to True for fine tuning

We keep all but the final layer of the model

And build our own classifier layer

forward() defines how the data will be passed through the model

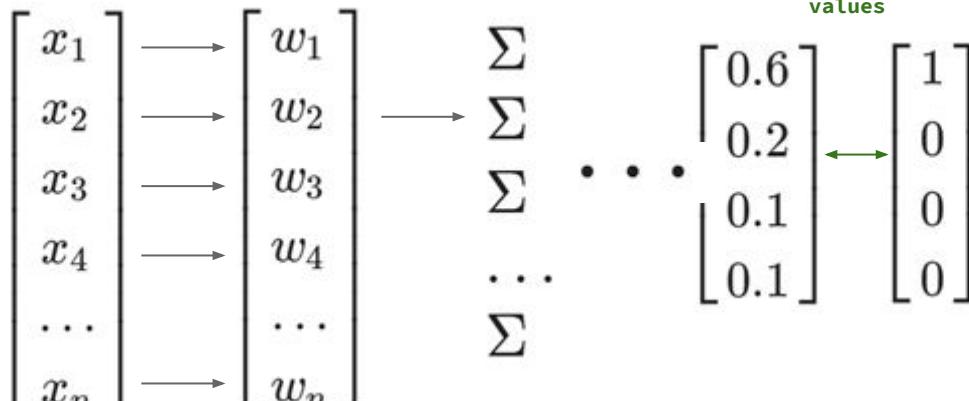
TRAINING AND TESTING



TRAINING A DEEP LEARNING MODEL



BATCH



OPTIMIZER:
updates the weights based on the
loss at the speed set by the
LEARNING RATE. Weights updated after
each BATCH

Hyperparameters

LOSS FUNCTION:
measures the
distance between the
predicted and actual
values

The model trains so that we
can get weights between the
layers

We set hyperparameters for
training

Some (loss, optimizer) are
often established rules of
thumb

Some (learning rate, batch
size, epoch) are found
through trial and error

TRAINING ON A GPU IN A 100 EASY STEPS...

Step 1: Choose an Amazon Machine Image (AMI)

Deep Learning AMI (Amazon Linux Version 2013.09) - ami-03dca0a7d3a4659

With this deep learning framework pre-installed: MPI4Py, TensorFlow, PyTorch, Keras, Chainer, Caffe2, Theano & DNN, configured with NVIDIA QDNN, iQDNN, NCCL, & Intel MKL-DNN. For a fully managed experience, check <https://aws.amazon.com/bagelar>

Deep Learning AMI (Amazon Linux Version 2013.09-2) - ami-077154a63632c295

With this deep learning framework pre-installed: MPI4Py, TensorFlow, PyTorch, Keras, Chainer, Caffe2, Theano & DNN, configured with NVIDIA QDNN, iQDNN, NCCL, & Intel MKL-DNN. For a fully managed experience, check <https://aws.amazon.com/bagelar>

Deep Learning AMI (Amazon Linux Version 2013.09-3) - ami-077154a63632c295

With this deep learning framework pre-installed: MPI4Py, TensorFlow, PyTorch, Keras, Chainer, Caffe2, Theano & DNN, configured with NVIDIA QDNN, iQDNN, NCCL, & Intel MKL-DNN. For a fully managed experience, check <https://aws.amazon.com/bagelar>

Deep Learning AMI (Amazon Linux Version 2013.09-4) - ami-077154a63632c295

With this deep learning framework pre-installed: MPI4Py, TensorFlow, PyTorch, Keras, Chainer, Caffe2, Theano & DNN, configured with NVIDIA QDNN, iQDNN, NCCL, & Intel MKL-DNN. For a fully managed experience, check <https://aws.amazon.com/bagelar>

Deep Learning AMI (Amazon Linux Version 2013.09-5) - ami-077154a63632c295

With this deep learning framework pre-installed: MPI4Py, TensorFlow, PyTorch, Keras, Chainer, Caffe2, Theano & DNN, configured with NVIDIA QDNN, iQDNN, NCCL, & Intel MKL-DNN. For a fully managed experience, check <https://aws.amazon.com/bagelar>

Deep Learning AMI (Amazon Linux Version 2013.09-6) - ami-077154a63632c295

With this deep learning framework pre-installed: MPI4Py, TensorFlow, PyTorch, Keras, Chainer, Caffe2, Theano & DNN, configured with NVIDIA QDNN, iQDNN, NCCL, & Intel MKL-DNN. For a fully managed experience, check <https://aws.amazon.com/bagelar>

Deep Learning AMI (Amazon Linux Version 2013.09-7) - ami-077154a63632c295

With this deep learning framework pre-installed: MPI4Py, TensorFlow, PyTorch, Keras, Chainer, Caffe2, Theano & DNN, configured with NVIDIA QDNN, iQDNN, NCCL, & Intel MKL-DNN. For a fully managed experience, check <https://aws.amazon.com/bagelar>

Deep Learning AMI (Amazon Linux Version 2013.09-8) - ami-077154a63632c295

With this deep learning framework pre-installed: MPI4Py, TensorFlow, PyTorch, Keras, Chainer, Caffe2, Theano & DNN, configured with NVIDIA QDNN, iQDNN, NCCL, & Intel MKL-DNN. For a fully managed experience, check <https://aws.amazon.com/bagelar>

Deep Learning AMI (Amazon Linux Version 2013.09-9) - ami-077154a63632c295

With this deep learning framework pre-installed: MPI4Py, TensorFlow, PyTorch, Keras, Chainer, Caffe2, Theano & DNN, configured with NVIDIA QDNN, iQDNN, NCCL, & Intel MKL-DNN. For a fully managed experience, check <https://aws.amazon.com/bagelar>

Deep Learning AMI (Amazon Linux Version 2013.09-10) - ami-077154a63632c295

With this deep learning framework pre-installed: MPI4Py, TensorFlow, PyTorch, Keras, Chainer, Caffe2, Theano & DNN, configured with NVIDIA QDNN, iQDNN, NCCL, & Intel MKL-DNN. For a fully managed experience, check <https://aws.amazon.com/bagelar>

Choose an Image

Step 2: Choose an Instance Type

Compute Optimized (c1.4xlarge) Select

Compute Optimized (c1.8xlarge) Select

Compute Optimized (c3.4xlarge) Select

Compute Optimized (c3.8xlarge) Select

Compute Optimized (c4.4xlarge) Select

Compute Optimized (c4.8xlarge) Select

Compute Optimized (g2.2xlarge) Select

Compute Optimized (g2.8xlarge) Select

Compute Optimized (g3.4xlarge) Select

Compute Optimized (g3.8xlarge) Select

Compute Optimized (g3.16xlarge) Select

Compute Optimized (g3.32xlarge) Select

GPU Instances (p2.8xlarge) Select

GPU Instances (p2.12xlarge) Select

GPU Instances (p2.16xlarge) Select

GPU Instances (p2.24xlarge) Select

GPU Instances (p2.32xlarge) Select

GPU Instances (p3.2xlarge) Select

GPU Instances (p3.8xlarge) Select

GPU Instances (p3.16xlarge) Select

GPU Instances (p3.32xlarge) Select

GPU Instances (p3.64xlarge) Select

Choose your compute option

```
# [root@ip-172-31-41-10 ~]# curl https://aws.amazon.com/lambda/api/v2015-03-31/functions/function/arn:aws:lambda:us-east-1:123456789012:function:myfunction/config
{
  "FunctionName": "myfunction",
  "Runtime": "nodejs10.x",
  "Handler": "index.handler",
  "Role": "arn:aws:iam::123456789012:role/lambdaBasicExecutionRole",
  "Code": {
    "ZipFile": "-----"
  },
  "MemorySize": 128,
  "Timeout": 3,
  "Version": "1",
  "TracingConfig": {
    "Mode": "None"
  },
  "Environment": {
    "Variables": {
      "AWS_LAMBDA_FUNCTION_NAME": "myfunction"
    }
  },
  "DeadLetterConfig": {
    "TargetArn": "arn:aws:lambda:us-east-1:123456789012:queue/lambda-dlq"
  },
  "EnvironmentVariables": {
    "AWS_LAMBDA_FUNCTION_NAME": "myfunction"
  },
  "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction"
}

# [root@ip-172-31-41-10 ~]# curl https://aws.amazon.com/lambda/api/v2015-03-31/functions/function/arn:aws:lambda:us-east-1:123456789012:function:myfunction/config
{
  "FunctionName": "myfunction",
  "Runtime": "nodejs10.x",
  "Handler": "index.handler",
  "Role": "arn:aws:iam::123456789012:role/lambdaBasicExecutionRole",
  "Code": {
    "ZipFile": "-----"
  },
  "MemorySize": 128,
  "Timeout": 3,
  "Version": "1",
  "TracingConfig": {
    "Mode": "None"
  },
  "Environment": {
    "Variables": {
      "AWS_LAMBDA_FUNCTION_NAME": "myfunction"
    }
  },
  "DeadLetterConfig": {
    "TargetArn": "arn:aws:lambda:us-east-1:123456789012:queue/lambda-dlq"
  },
  "EnvironmentVariables": {
    "AWS_LAMBDA_FUNCTION_NAME": "myfunction"
  },
  "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction"
}

# [root@ip-172-31-41-10 ~]# curl https://aws.amazon.com/lambda/api/v2015-03-31/functions/function/arn:aws:lambda:us-east-1:123456789012:function:myfunction/config
{
  "FunctionName": "myfunction",
  "Runtime": "nodejs10.x",
  "Handler": "index.handler",
  "Role": "arn:aws:iam::123456789012:role/lambdaBasicExecutionRole",
  "Code": {
    "ZipFile": "-----"
  },
  "MemorySize": 128,
  "Timeout": 3,
  "Version": "1",
  "TracingConfig": {
    "Mode": "None"
  },
  "Environment": {
    "Variables": {
      "AWS_LAMBDA_FUNCTION_NAME": "myfunction"
    }
  },
  "DeadLetterConfig": {
    "TargetArn": "arn:aws:lambda:us-east-1:123456789012:queue/lambda-dlq"
  },
  "EnvironmentVariables": {
    "AWS_LAMBDA_FUNCTION_NAME": "myfunction"
  },
  "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction"
}

# [root@ip-172-31-41-10 ~]# curl https://aws.amazon.com/lambda/api/v2015-03-31/functions/function/arn:aws:lambda:us-east-1:123456789012:function:myfunction/config
{
  "FunctionName": "myfunction",
  "Runtime": "nodejs10.x",
  "Handler": "index.handler",
  "Role": "arn:aws:iam::123456789012:role/lambdaBasicExecutionRole",
  "Code": {
    "ZipFile": "-----"
  },
  "MemorySize": 128,
  "Timeout": 3,
  "Version": "1",
  "TracingConfig": {
    "Mode": "None"
  },
  "Environment": {
    "Variables": {
      "AWS_LAMBDA_FUNCTION_NAME": "myfunction"
    }
  },
  "DeadLetterConfig": {
    "TargetArn": "arn:aws:lambda:us-east-1:123456789012:queue/lambda-dlq"
  },
  "EnvironmentVariables": {
    "AWS_LAMBDA_FUNCTION_NAME": "myfunction"
  },
  "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction"
}

# [root@ip-172-31-41-10 ~]# curl https://aws.amazon.com/lambda/api/v2015-03-31/functions/function/arn:aws:lambda:us-east-1:123456789012:function:myfunction/config
{
  "FunctionName": "myfunction",
  "Runtime": "nodejs10.x",
  "Handler": "index.handler",
  "Role": "arn:aws:iam::123456789012:role/lambdaBasicExecutionRole",
  "Code": {
    "ZipFile": "-----"
  },
  "MemorySize": 128,
  "Timeout": 3,
  "Version": "1",
  "TracingConfig": {
    "Mode": "None"
  },
  "Environment": {
    "Variables": {
      "AWS_LAMBDA_FUNCTION_NAME": "myfunction"
    }
  },
  "DeadLetterConfig": {
    "TargetArn": "arn:aws:lambda:us-east-1:123456789012:queue/lambda-dlq"
  },
  "EnvironmentVariables": {
    "AWS_LAMBDA_FUNCTION_NAME": "myfunction"
  },
  "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction"
}
```

Step 7: Review Instance Launch

Your instance security group, launch record, S3, is set to the world.

Your instance security group is set to the world. If you're not sure, we recommend that you update your security group rules to allow access from known IP addresses only. You can also open additional ports in your security group to facilitate access to the application or service you're running, e.g., HTTP (80) for web servers. [Edit security group](#)

Your instance configuration is not eligible for the free usage tier. To learn if an instance's configuration is eligible for the free usage tier, choose your AMI, security group, instance type, configuration options, or storage device. Learn more about free usage for eligible and usage restrictions. [Don't show me this again](#)

AM Details

Deep Learning AMI (Amazon Linux Version 2013.09-10) - ami-077154a63632c295

With this deep learning framework pre-installed: MPI4Py, TensorFlow, PyTorch, Keras, Chainer, Caffe2, Theano & DNN, configured with NVIDIA QDNN, iQDNN, NCCL, & Intel MKL-DNN. For a fully managed experience, check <https://aws.amazon.com/bagelar>

Instance Type

Instance Type	EC2	vCPUs	Memory (GB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
64xlarge	11.70	4	81	ESS only	Yes	High

Security Groups

Request an increase...wait

Request an increase...wait

Spend an hour configuring instance

Training on a GPU can offer significant speed benefits

Signing up is hard to do – so is setting up an instance

Training a model on a cloud is not a cheap option

My experience was mixed but I would try it again

TRAINING IN PYTORCH

```
for epoch in n_epochs:
```

```
    for batch in train_set:  
  
        run model on images  
        return predictions  
        compare against actual  
        calculate loss  
        calculate accuracy  
        store accuracy and loss  
        update parameters
```

```
    for batch in val_set:  
  
        run model on images  
        return predictions  
        compare against actual  
        calculate loss  
        calculate accuracy  
        store accuracy and loss
```

```
    calculate epoch score
```

```
return final parameters and history object
```

```
for batch_i, (inputs, labels) in enumerate(train_loader):  
  
    if train_on_gpu:  
        inputs, labels = inputs.cuda(), labels.cuda()  
        model.cuda()  
  
    model.train()  
  
    optimizer.zero_grad()  
  
    outputs = model(inputs)  
  
    _, predictions = torch.max(outputs, 1)  
  
    correct_predictions = torch.sum(predictions ==  
labels).item()  
  
    loss = criterion(outputs, labels)  
  
    loss.backward()  
  
    loss_per_batch = loss.item()  
  
    optimizer.step()
```

IT'S COMPLICATED...

TRAINING THE MODEL



Not a classic training curve...

...and not SOTA



Training should test different combinations of hyperparameters

Difficult to initially develop a structured approach to training

I found training unstable and hard to understand what was driving the result

MODEL PERFORMANCE

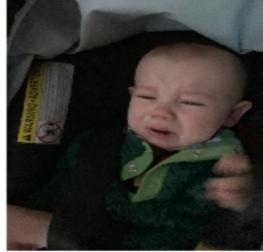
Actual: Feed | Predicted: Feed



Actual: Happy | Predicted: Happy



Actual: Upset | Predicted: Sleep



Actual: Feed | Predicted: Upset



Overall performance was better than guessing but far from state of the art

Difficult challenge to learn differences within the same face

More, and better, data would be the biggest improvement I could make

```
# load test train set  
test_vb8e100ol2g.load_state_dict(torch.load('saved_models/vb8e100ol2g.pt', map_location='cpu'))
```

NEXT STEPS AND LESSONS LEARNED



NEXT STEPS

Improve HarperNet



Add data
Optimise training
Deploy on mobile

Image captioning



“Harper is crawling down
the stairs!”

Train LSTM model to
caption images and
provide alerts

NLP



Build model to
understand and
translate speech

Chatbot



Build accessible
voice repository
of games and ideas

KEY LESSONS

ALL ABOUT THE DATA

LEARNING THROUGH OWN PROJECTS

D.I.Y. DEEP LEARNING

PYTORCH

QUESTIONS?