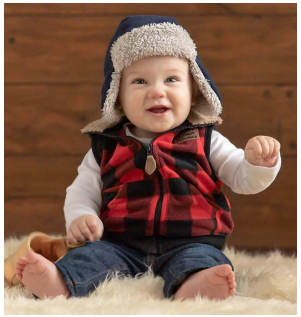
Predicting my son's mood with deep learning

Dan Howarth

Neural networks are hard to train...

New Parents ~
no labelled data



Neural Network a.k.a.
Harper

?

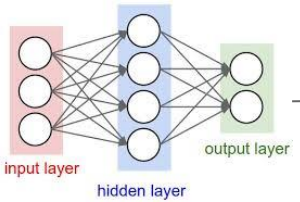
Grandparents ~
data old and corrupted

...hopefully Artificial ones are a bit easier

My aim was to build a predictive model...

DATA

MODEL



TRAINING AND TESTING

Upset

Sleep

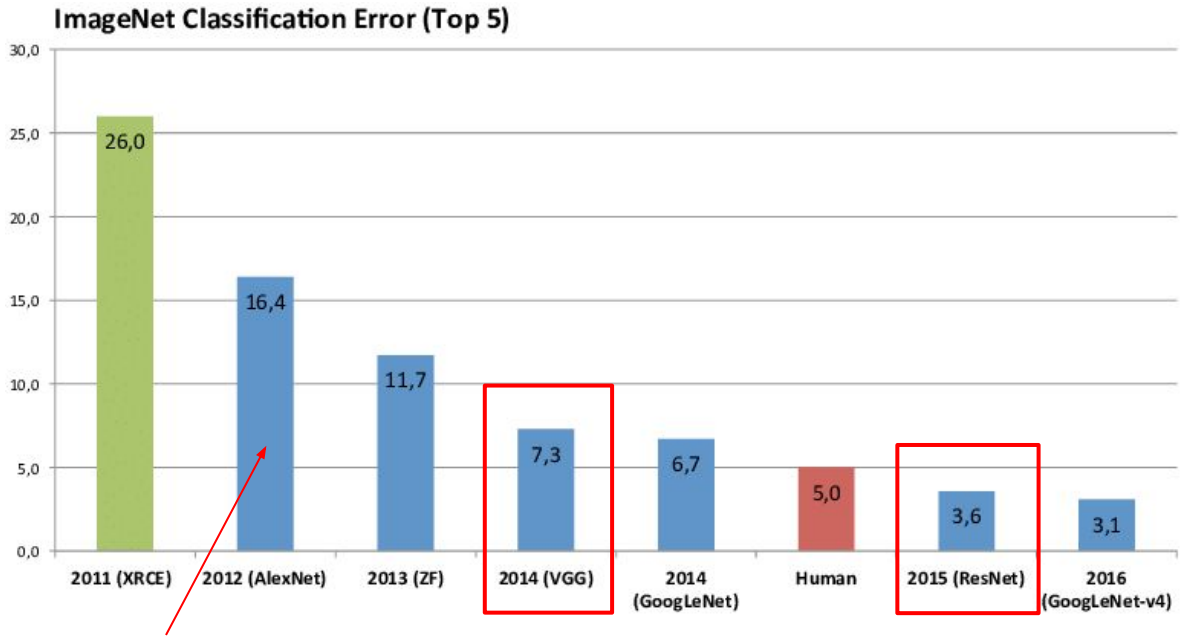
Happy

Feed

PyTorch

...and learn more about computer vision in the process

What is computer vision?



ImageNet scores to 2016

First Deep Neural Network winner

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

https://www.researchgate.net/figure/Winner-results-of-the-ImageNet-large-scale-visual-recognition-challenge-LSVRC-of-the_fig7_324476862


DATA

What makes a good dataset?

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







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

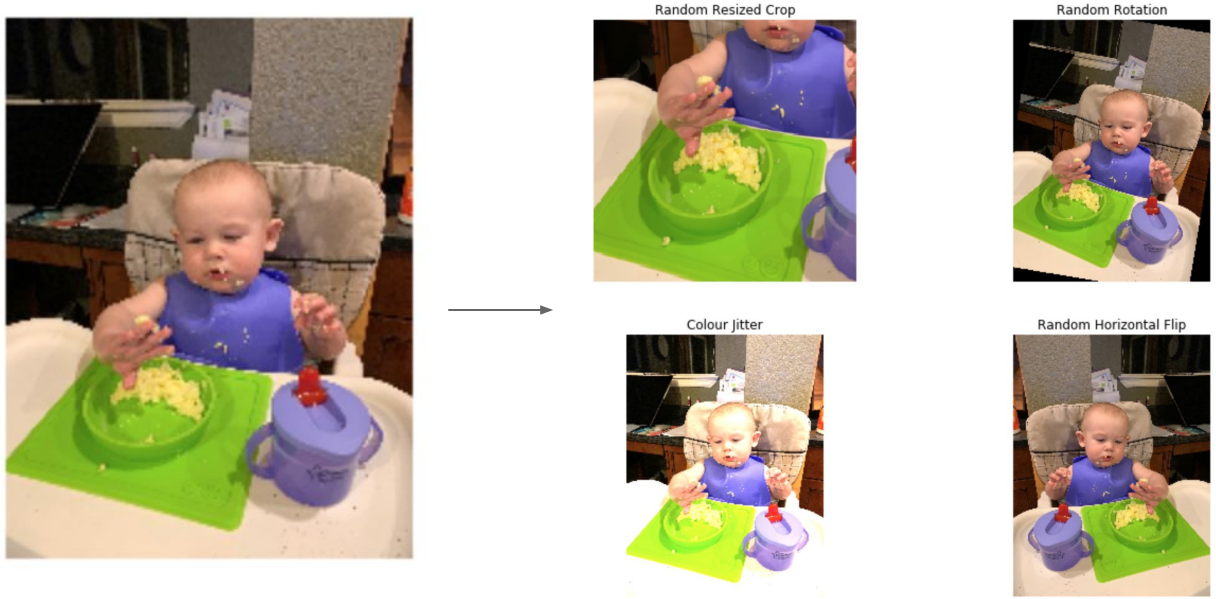
Representative

Varied

Balanced classes

Different train, val and test sets

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

HarperNet: instantiating a dataset

```
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)
```

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

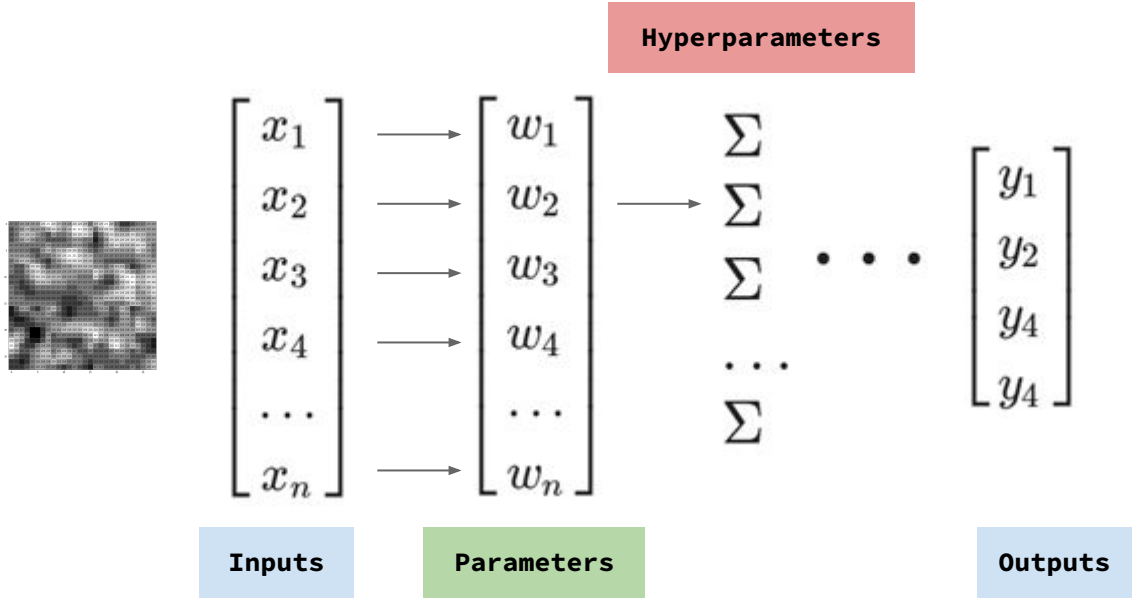
```
# train transforms
train_transform = transforms.Compose([transforms.ToPILImage(),
                                      transforms.RandomResizedCrop(224),
                                       transforms.RandomRotation(degrees=15),
                                       transforms.ColorJitter(),
                                       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 are an art as much as a science

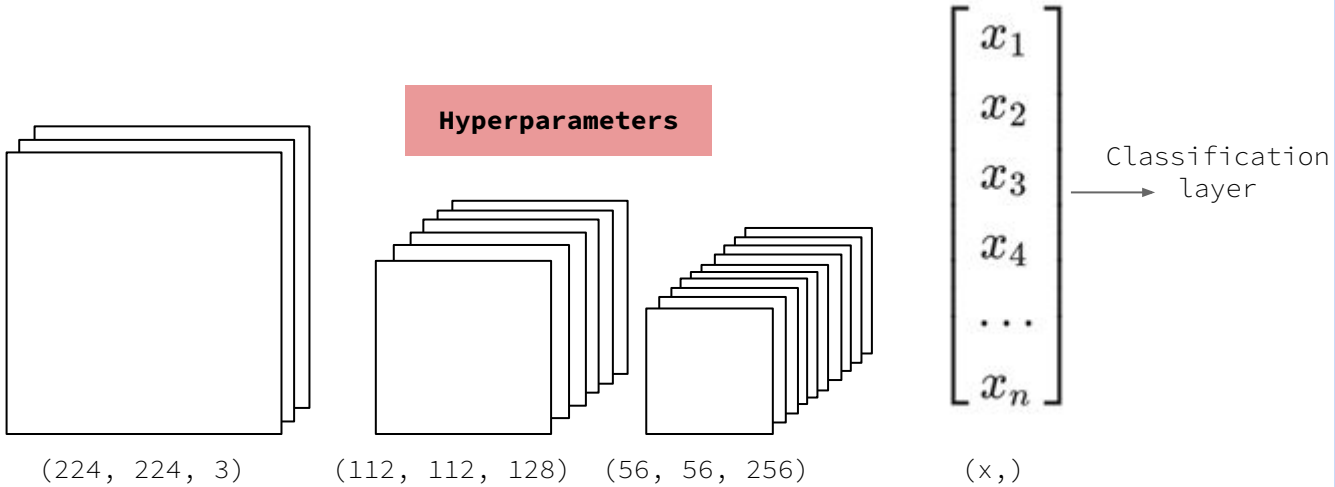
Deep learning models the relationships between inputs and outputs

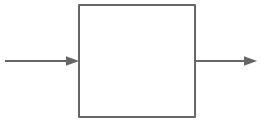
Hyperparameters include the number and size of layers and the activation functions

Parameters we train are the weights in between layers

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

WHY ARE CNNS USED FOR COMPUTER VISION?





Learned Convolutional Filter

Parameters

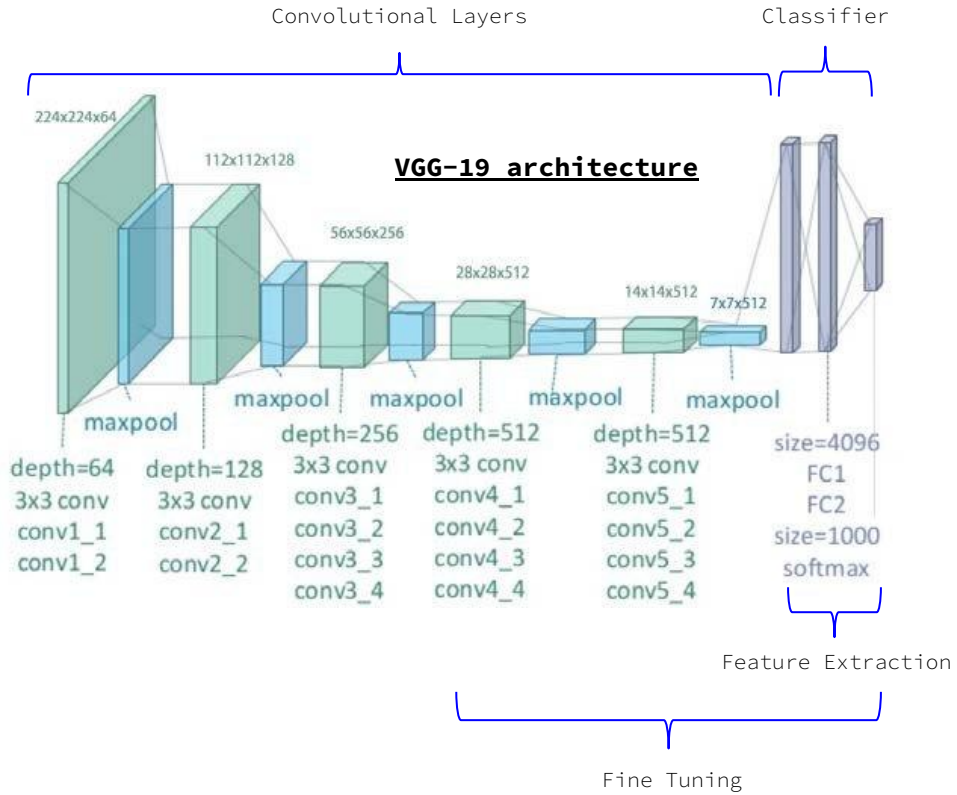
CNNs produce a better representation of the data before we go to the classifier stage

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?



https://www.researchgate.net/figure/llustration-of-the-network-architecture-of-VGG-19-model-conv-means-convolution-FC-means_fig2_325137356

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

```python
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

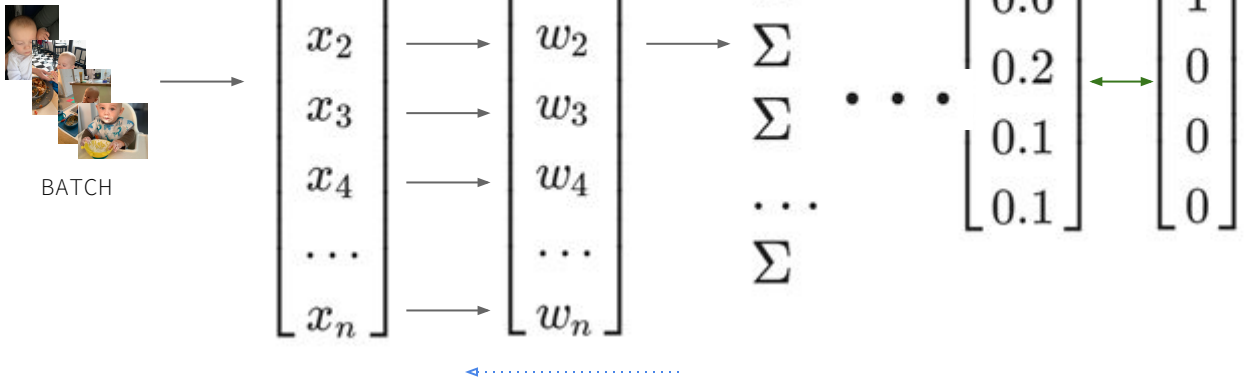
We keep all but the final layer of the model

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



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

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

Hyperparameters

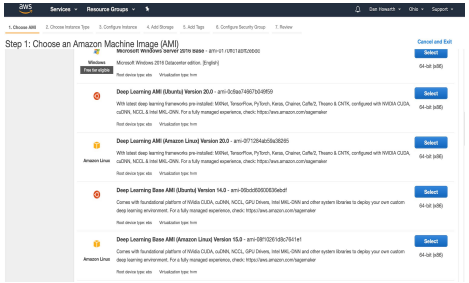
The model trains so that we can get weights between the layers

We set hyperparameters for training

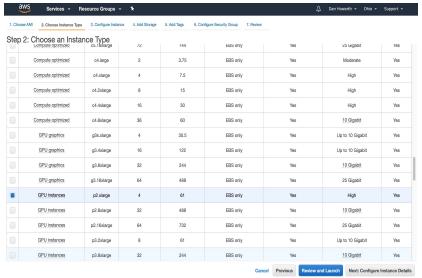
Some (loss, optimizer) are often established rules of thumb

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

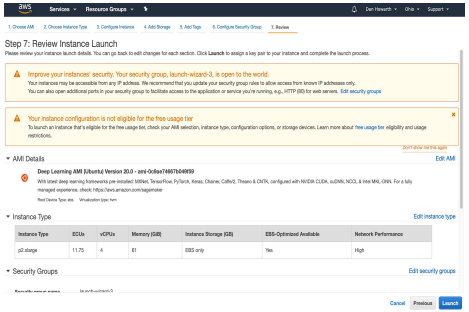
Training on a GPU in a 100 ~~EASY~~ steps...



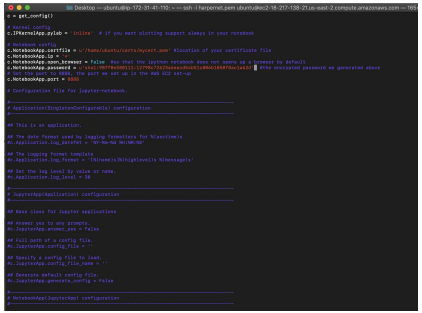
Choose an Image



Choose your compute option



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
```

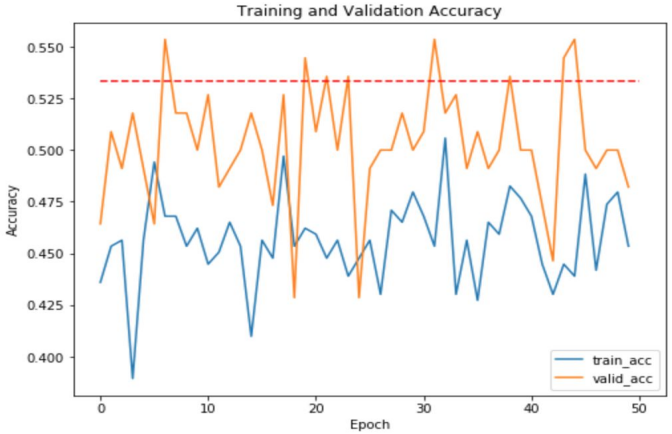
It's complicated...

```
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()
```

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

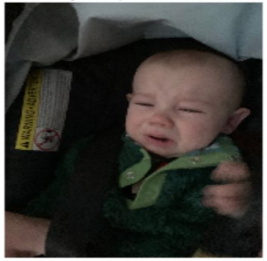
I found training unstable and hard to understand what was driving the result

Model performance

Actual: Feed | Predicted: Feed

Actual: Happy | Predicted: Happy

Actual: Upset | Predicted: Sleep



Actual: Feed | Predicted: Upset

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

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

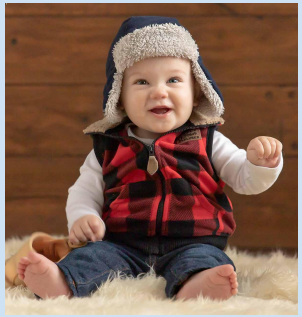
Difficult challenge to learn differences within the same face

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

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?