## Repurposing neural networks

An introduction to transfer learning Dr Jason Rigby
$11^{\text {th }}$ Apr 2018

This presentation will cover:

- A speedy introduction to dense and convolutional neural networks (CNN)
- An analysis of what the layers in a CNN "see"
- A case study where a pretrained CNN is repurposed for a different domain
- A case study where a pretrained CNN is used to generate images


## The fundamentals

## The fundamentals

Logistic regression

Logistic regression maps values from [-infinity, +infinity] to a fixed range of $[\mathbf{0}, \mathbf{1}]$ in such a way that can be interpreted as a probability.

The prediction of a logistic regression model is binary; it models the probability of belonging to a single class.

$$
\begin{aligned}
& \sigma(t)=\frac{1}{1+e^{-t}} \\
& t=\log \left(\frac{p}{1-p}\right) \approx w_{0}+w_{1} x_{1}+\cdots+w_{n} x_{n}=\mathbf{w}^{T} \mathbf{x}
\end{aligned}
$$



## The fundamentals

## Multi-class logistic regression

Multi-class logistic regression (softmax regression) normalises the predictions of multiple logistic regressions such that the sum of each regression is 1 .

It models the probability of an belonging to one of several classes.

$$
\begin{aligned}
& \sigma(t)_{k}=\frac{e^{t_{k}}}{\sum_{i=1}^{K} e^{t_{i}}} \\
& t=\log \left(\frac{p}{1-p}\right) \approx w_{0}+w_{1} x_{1}+\cdots+w_{n} x_{n}=\mathbf{w}^{T} \mathbf{x}
\end{aligned}
$$

## The fundamentals

Multi-class logistic regression

$$
\sigma(t)_{k}=\frac{e^{t_{k}}}{\sum_{i=1}^{K} e^{t_{i}}} \quad \begin{aligned}
& \text { A set of weights corresponding to each pixel exist for each digit (0-9) that } \\
& \text { predict the probability of a given image being a given digit. }
\end{aligned}
$$

$t=\log \left(\frac{p}{1-p}\right) \approx w_{0}+w_{1} x_{1}+\cdots+w_{n} x_{n}=\mathbf{w}^{T} \mathbf{x}$

1: [0, 1,0,0,0,0,0,0,0,0.]

$0:[1,0,0,0,0,0,0,0,0,0$.

$6:[0,0,0,0,0,0,1,0,0,0$.


6: [0, 0, 0, 0, 0, 0, 1, 0, 0, 0.]


## The fundamentals

Multi-class logistic regression


The weights look like this! White indicates a positively for a given digit, black is a negative indication


## The fundamentals

## Optimisation

The parameters $w$ can be optimised using gradient descent; this involves incrementally adjusting $w$ by examining their derivative with respect to an error function (loss function, $J$ ) to make the error as small as possible.
$\mathbf{w}_{t+1}=\mathbf{w}_{t}-\eta \frac{\partial J}{\partial \mathbf{w}}$

## The fundamentals

Logistic regressions can be stacked together to account for more complex relationships


The weights are adjusted iteratively to minimise an error function, which quantifies the deviation of the estimated output from the expected output

Non-linearities are essential; they prevent the network from simplifying to a simple linear regression.

See:
https://en.wikipedia.org/wiki/Universal_approximation_theorem
Nodes represen

## The fundamentals

There are many non-linearities to choose from






Sigmoid is a non-negotiable for the output of networks predicting probabilities, but ReLU is more common for the middle layers these days.

## The fundamentals

Dense networks are problematic for images


The weights are adjusted iteratively to minimise an
error function, which quantifies the deviation of the estimated output from the expected output

As more layers and bigger images are analysed, the number of weights increases rapidly.

## The fundamentals

The convolution operation

$(0 \times 0)+(-1 \times 0)+(4 \times 0)+$ $(3 \times-1)+(-3 \times 2)+(-3 \times-1)+$
$(2 \times 0)+(-1 \times 0)+(0 \times 0)=-6$


Convolutions apply a small set of weights to the entire image as a weighted sum. Traditionally, convolutions were hand-crafted for embossing or edge detection; neural networks learn these parameters automatically.

## The fundamentals

The max pooling operation



Convolutions apply a small set of weights to the entire image as a weighted sum. Traditionally, convolutions were hand-crafted for embossing or edge detection; neural networks learn these parameters automatically.

## The fundamentals

Common CNN architectures


Most CNN networks consist of successive convolution and pooling operations, followed by dense layers for classification

## Using VGG-19

## Using VGG-19

## Some background



Developed by the Visual Geometry Group, Oxford University in 2015
Achieved about a $71 \%$ top-1 accuracy, and about $90 \%$ for top-5 accuracy in the ImageNet classification competition (validation set of 150,000 images)

Designed to process images of $224 \times 224$ pixels, outputs probabilities for 1000 different classes

Has about 143,000,000 trainable parameters
Original paper: https://arxiv.org/pdf/1409.1556.pdf

## Using VGG-19

Using the network

In bold is my annotation; title below is the network prediction from ImageNet classes; percentage probability given in parentheses.



## Using VGG-19

What does the network see?


## Using VGG-19

What does the network see?


## Using VGG-19

What does the network see?


## Using VGG-19

What does the network see?

F

## Using VGG-19

What does the network see?


## Using VGG-19

What does the network see?

Rather than activate filters using an image, we can generate an image designed to activate specific filters.
https://github.com/jasonrig/transfer-learning-demos
python3 -m TransferLearningDemo.demos.vgg_19_activate_filters

## VGG-19

## What does the network see?



Filters closer to the input capture lower-level features

Repurposing VGG-19

## Repurposing VGG-19

## Parameter reuse

These features generally don't change much


## Repurposing VGG-19

## Parameter reuse

These features generally don't change much


## Motivation:

Pretrained networks have learnt parameters from a vast number of training exampled (ImageNet +14 mil images)
Some domains have very limited data

Low-level features in CNNs are almost universal; always hard edges or simple textures

## Repurposing VGG-19

Retinal haemorrhage detection - STARE dataset


## Dataset description:

400 retinal photographs
44 features annotates per image
Each feature can have more than one classification, e.g. microaneurisms can be:

- Unknown
- Absent
- Few anywhere
- Many anywhere

VGG-19 has been retrained as a binary classifier for retinal haemorrhages with approximately 75-80\% accuracy

Training took approximately 20 minutes on a single GPU

## Repurposing VGG-19

Retinal haemorrhage detection - Example results on unseen images
Testing the VGG-19 haemorrhage detector:


https://clinicalgate.com/nonproliferative-diabetic-retinopathy-and-diabetic-macular-edema/
https://github.com/iasonrig/transfer-learning-demos
python3 -m TransferLearningDemo.demos.vgg_19_retrain_fc train python3 -m TransferLearningDemo.demos.vgg_19_retrain_fc evaluate python3 -m TransferLearningDemo.demos.vgg_19_retrain_fc predict <image_name>

## Repurposing VGG-19

## Retinal haemorrhage detection - Training methodology

STARE images were partitioned into:

- Haemorrhagic [0,1]
- Non-haemorrhagic [1, 0]

All conv layers were frozen; their parameters remained unchanged during training

All fc layers were allowed to train

The final fc layer with 1000 outputs was replaced with a new one with 2 outputs, reflecting the new labels and initialised to random weights

75 images were selected randomly for testing and not included during training


## Repurposing VGG-19

## Retinal haemorrhage detection - Training methodology

## Data augmentation:

Necessary because there are only 325 training examples
Each time an image was presented to the model for training, it was:

- Randomly flipped left-right
- Randomly flipped top-bottom
- Randomly cropped (cropped patches varied from $224 \times 224$ - no cropping - to $200 \times 200$ )
- Resized to $224 \times 224$

Data augmentation allows us to anticipate and produce variations that might be seen outside of our training set without obtaining new data

## Reinterpreting VGG-19

## Reinterpreting VGG-19

Neural style transfer

The activations generated by pretrained CNN models can be used to generate new images in the style of other images

This requires us to optimise for:

- Image content; and,
- Image style

https://arxiv.org/pdf/1508.06576.pdf


## Reinterpreting VGG-19

## Neural style transfer - defining style

Style is defined using the Gram matrix (similar to the covariance matrix) between the activation maps of each filter in each layer:

$$
G_{A}=A \cdot A^{T}
$$

Where $A$ is a 2D matrix containing an unrolled vector of each activation for each filter in the given network layer

Intuitively, it captures the distribution of activations, which reflect now likely certain structures of the image occur together

To optimise for style, we essentially minimise the squared difference between $G$ of the generated image and the style image for selected network layers:

$$
J_{S}=\gamma \sum_{i=1}^{n_{c}} \sum_{j=1}^{n_{c}}\left(G_{i j}^{\text {style }}-G_{i j}^{\text {generated }}\right)^{2}
$$

## Reinterpreting VGG-19

Neural style transfer - defining content
Content is defined as the difference between network activations of the content image and the generated image.

$$
J_{c}=\gamma \sum\left(A^{\text {content }}-A^{\text {generated }}\right)^{2}
$$

The total loss is therefore:

$$
J=\alpha J_{S}+(1-\alpha) J_{C}
$$

https://github.com/jasonrig/transfer-learning-demos
python3 -m TransferLearningDemo.demos.vgg_19_style_transfer

## The code

Code used to produce all figures available on GitHub https://github.com/jasonrig/transfer-learning-demos

Pip installable! GPU not required (except for VGG-19 retraining)
pip install -U git+https://github.com/jasonrig/transfer-learning-demos

Pretrained model data from:
https://github.com/tensorflow/models/tree/master/research/slim


