Probabilistic Head Pose Estimation

Raymond Allen
MSc Data Science & Analytics
105537797

Thesis submitted for the degree of
Master of Science

NATIONAL UNIVERSITY OF IRELAND, CORK

Faculty of Science

Department of Computer Science

26 August 2019

Head of Department: Prof Barry O'Sullivan

Supervisor: Dr Gregory Provan

Contents

List of Figures ii
List of Tables iii
List of Abbreviations iv
Acknowledgements vi
Abstract vii

1 Introduction **1**
1.1 Aims of this work 2

2 Related Work **3**

3 Probabilistic Model **5**
3.1 Vector Representation 5
3.2 Von Mises-Fisher Distribution 6
3.3 Finite Mixture of von Mises-Fisher Distributions 8
3.4 Advantage of Single Vector Approach 9
3.5 Point Prediction 10

4 Experiments **11**
4.1 Data and Implementation Details 11
4.2 Augmentation 11
4.3 Number of Mixture Components 12
4.4 Networks and Resolutions 12
4.5 Discussion of Results 15

5 Conclusion and Future Work **20**

A Network Architectures **26**
A.1 Note on Activation Functions 26
A.2 LeNet-5 27
A.3 VGG-16 27
A.4 ResNet-34 27
A.5 GoogLeNet 28
A.6 Inception-v3 30

B Technical Specifications **34**

List of Figures

3.1 The Euler angles 5
3.2 Parameters and shapes of von Mises and von Mises-Fisher (3-dimensional case) distributions [Kva19]. 7
3.3 An ambiguously oriented face 8

4.1 Augmentation techniques used. 12
4.2 Validation set loss using different augmentation techniques (VGG network). 13
4.3 Validation set loss by number of mixture components (VGG network). 14
4.4 Validation set performance by network, input resolution 32x32 pixels. 16
4.5 Validation set performance by network, input resolution 64x64 pixels. 17
4.6 Validation set performance by network, input resolution 112x112 pixels (149x149 pixels for Inception-v3). 18
4.7 Validation set loss (negative log-likelihood) chart by input resolution for each network. 19

A.1 LeNet-5 27
A.2 VGG-16. $^{*(n)}$: Layer included only if input resolution $\geq n$. . . . 28
A.3 Residual blocks as featured in ResNet-34. 29
A.4 ResNet-34. $^{*(n)}$: Layer included only if input resolution $\geq n$ for pooling layer, replaced by identity block if reduction block. . . 29
A.5 Inception Module as used in GoogLeNet. Filter numbers vary per individual module. 30
A.6 GoogLeNet. $^{*(n)}$: Layer included only if input resolution $\geq n$. . . 31
A.7 A convolution block as used in Inception-v3, consisting of a convolution layer (variable filter size) followed by a batch normalization layer. 32
A.8 Inception Module C as used in Inception-v3. Filter numbers vary per individual module. 32
A.9 Inception Reduction Module B as used in Inception-v3. 32
A.10 Inception-v3. $^{*(n)}$: Layer included only if input resolution $\geq n$. . 33

List of Tables

4.1 Results for input resolution 32x32 pixels. 14
4.2 Results for input resolution 64x64 pixels. 14
4.3 Results for input resolution 112x112 pixels (149x149 pixels for Inception-v3). 14
4.4 Summary of log-likelihood results across all networks and input resolutions. 15
4.5 Summary of MAAE results across all networks and input resolutions. 15
4.6 Comparison of test set results with those of benchmarks on the AFLW dataset. 15

B.1 Technical Specifications 34

List of Abbreviations

AFLW Annotated Facial Landmarks in the Wild [KWRB11]

CNN Convolutional Neural Network

GPU Graphical Processing Unit

MAAE Mean Absolute Angular Error

MBR Minimum Bayes Risk

ReLU Rectified Linear Unit

VGG Visual Geometry Group [SZ14]

VMF Von Mises-Fisher

I, Raymond Allen, certify that this thesis is my own work and has not been submitted for another degree at University College Cork or elsewhere.

Raymond Allen

Acknowledgements

My sincere thanks to my supervisor, Dr. Gregory Provan, for his guidance and trust during the course of this project.

I would also like to thank the numerous professors and tutors of the Departments of Computer Science and Statistics who have contributed to my education this year, and to David O'Byrne and the other members of the Computer Science helpdesk for their assistance in my battles against the GPU.

Thanks finally to Michael Opitz of TU Graz for providing access to the excellent AFLW dataset that this project was built with.

Abstract

Head pose estimation is a well-studied problem in the field of computer vision, with its most obvious application being to autonomous driving systems. Almost all work to date attempts to generate a single prediction from an input image, either by classifying to a discrete category or outputting a continuous value for pose direction. A recent paper by Prokudin et. al. was among the first to recognise the importance of uncertainty estimation for this task, and developed a probabilistic model to predict pose direction with uncertainty for each of the three Euler angles. Here, we adapt their work, differing from theirs in two main ways: firstly, we use the von Mises-Fisher distribution on the sphere to predict a single gaze direction jointly across these angles, with uncertainty estimation. To the best of our knowledge this is the first paper to use this distribution for this task. Secondly, we use a finite mixture of distributions for our final model, rather than the infinite mixture favoured by Prokudin et. al. We then evaluate this model across a number of different network architectures and input resolutions on the AFLW dataset, while also investigating the usefulness of data augmentation techniques on controlling overfitting.

Chapter 1

Introduction

Today's most advanced autonomous vehicles are able to achieve near-full automation under highway driving conditions, but require human driver takeover in urban areas. One of the main reasons for this is the presence of highly unpredictable pedestrians in built-up areas. In order to increase the level of automation in these environments, it is necessary to be able to predict the likely actions of pedestrians in much the same way as a human driver would.

A key observation is that a pedestrian's head pose, or gaze direction, is an excellent predictor of their anticipated direction of motion. By training a model to detect head orientation from image data, we can provide an important input to a vehicle's decision-making framework.

Most work in this area has framed the problem either as a classification task – assigning each pedestrian image to one of 'up', 'down', 'slight left', or some such division of categories [SQLG15, TM15] – or, perhaps more naturally, a regression task, to predict a numerical value, or set of values, for the orientation [LGS19]. Other authors [EG10, FDGKG14] have made the observation that in addition to predicting such a value, a useful network should also produce an estimate of its confidence in its own prediction – we might ask how many accidents are avoided by human drivers braking when they are unsure whether they have seen a potential danger or not - and addressed this need by framing the problem probabilistically.

Like most recent work involving visual data, the most successful models to predict head pose have been based on deep convolutional neural networks (CNNs) [APK14, TM15]. While the results of these networks in an enormous array of image detection and analysis tasks speak for themselves, they do

provide a challenge in their parametrisation. How many layers should the network contain, of what type and how wide? What image resolution gives best results? Which training hyper-parameters should be chosen? Answers to some of these questions may be influenced by the constraints of the particular problem – for instance, large networks using high resolution images make severe demands on memory. In the context of autonomous driving, in which a network for head pose detection forms just one part of an enormously complex monitoring and predictive system, significant limitations on image size and CPU available are likely. Another such constraint is that pedestrian faces are likely to form a very small part of the visual field of a car's camera, so a network that can predict accurately from low resolution images is desirable.

1.1 Aims of this work

The aims of this work are:

1. To implement a probabilistic uncertainty-based deep convolutional neural network to predict head pose orientation from still image data of pedestrians. Unlike previous work, this project considers the combination of yaw, pitch and roll angles jointly to predict a gaze direction as a unit vector on the sphere, rather than predicting the angles separately.
2. To evaluate and compare the performance of this model using a range of standard network architectures and image resolutions, and, where possible, to compare results to that of previous work which used a simple point prediction.
3. To investigate the effect of data augmentation techniques on model training performance, and determine an optimal number of probability distribution mixture components to use.

Chapter 2

Related Work

Early work on head pose detection made use of hand-crafted features such as mean energy features [VDMK15], histograms of oriented gradients (HOGs) [FY11] or manifolds [LT12]. These features were used in combination with classifiers such as support vector machines (SVMs) [SM15], random forests [TK14, FDGKG14] or regressors such as linear regressors [GX14] or regression random forests [FGVG11]. Some of these models involved probabilistic methods [EG10, FDGKG14].

Convolutional neural networks, from early architectures such as LeNet [LJB+95] and AlexNet [KSH12] to more recent, more sophisticated forms such as VGGs [SZ14], Inception networks [SLJ+15] and residual networks (ResNets) [HZRS16, SIVA17] have consistently outperformed manually crafted feature models on image-based tasks such as handwriting detection and image recognition, so much so that they are now almost synonymous with such tasks. In contrast to earlier methods, these networks learn their own feature detection mechanisms in training, making them highly flexible and yet specialisable. Numerous authors have applied CNNs to head pose detection, often as a classification problem [SQLG15, TM15]. Others have taken the view, as this paper does, that posing the problem as a classification task limits the accuracy of the prediction and leads to information loss, and so have instead predicted a continuous-valued point estimate of pose direction [PC17, Rie18, APK14]. Liao et. al. [LGS19] defined a continuous-valued representation of orientation direction in terms of the 3 Euler angles on S^1.

However, treating the problem as a regression task means that one benefit of a classification framework – its simple conversion to a probabilistic output by

means of a softmax function on the output – is lost. Prokudin et. al. [PGN18], using the representation of Liao et. al., solved this apparent dilemma by modelling the predicted probability distribution of pose orientation as a mixture of von Mises distributions on the Euler angles, which the network is trained to predict the parameters of. Specifically, they show how a mixture model can be learned using either a finite or an infinite number of mixture components, the latter leading to distributions of arbitrary shape.

Interestingly, although the Liao [LGS19] paper defined spherical regression on general n-spheres, and applied regression on S^2 to the problem of surface normal estimation, they did not consider representing pose orientation as a vector on S^2 and regressing accordingly, as this paper does.

The success of neural networks is highly dependent on the data on which they are trained – more data is always better. In practice, adequate quality data is generally in limited supply. Data augmentation techniques such as shifting, flipping and rotating artificially extend a dataset [MG18], helping to prevent overfitting and make learning more robust. Another such technique, Cutout [DT17] has been shown to improve accuracy in top-performing image classifier networks such as that of Pham et. al. [PGZ+18].

Chapter 3

Probabilistic Model

This section initially follows the work of Prokudin et. al. [PGN18], before diverging, as will be explained.

3.1 Vector Representation

Annotated head pose data typically takes the form of a triple of angles known as the Euler angles. The individual angles have various names in different contexts: rotation around the vertical z-axis can be called the yaw, pan, or azimuthal angle, rotations around the left-right horizontal y-angle can be called pitch or elevation, while rotations around the forward-backward horizontal x-angle can be called roll or tilt. We shall use the yaw/pitch/roll terminology, denoting them as ϕ, θ and ψ.

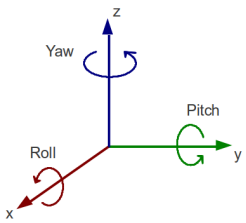


Figure 3.1: The Euler angles

Representing these angles in the obvious way leads to discontinuity problems. In degrees, a deviation of 1 degree either side of 0 is represented as either 1 degree or 359 degrees; yet these should result in the same value of the loss function. Representing angles 180 to 360 degrees as -180 to -0 only shifts the discontinuity to +/- 180. Taking the absolute value of the difference might seem

to prevent this issue, but leads to a non-differentiable loss function.

Prokudin et. al., following Liao et. al. [LGS19], overcome this difficulty by representing the angle as a 'Biternion' - a somewhat grandiose name for a (cos, sin) unit vector on the circle. Such a pair is easily predicted by a network by normalising a pair of output values:

$$f_{BT}(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \frac{\mathbf{W}\mathbf{x} + \mathbf{b}}{||\mathbf{W}\mathbf{x} + \mathbf{b}||}$$

A simple cosine distance is then used as the loss function:

$$L_{cos}(\mathbf{y_{pred}}, \mathbf{y_{true}}) = 1 - \frac{\mathbf{y_{pred}} \cdot \mathbf{y_{true}}}{||\mathbf{y_{pred}}|| \cdot ||\mathbf{y_{true}}||} = 1 - \mathbf{y_{pred}} \cdot \mathbf{y_{true}},$$

since $||\mathbf{y}|| = 1$.

Their approach involved using this representation to predict and calculate a loss for each of the three angles separately. In this work, however, we take a different approach. Instead of predicting three unit vectors on the circle, we predict a single unit vector on the sphere, representing gaze direction. To convert from our Euler angles to this unit vector, we use the identities:

$$x = \cos(\phi).\cos(\theta)$$

$$y = \sin(\phi).\cos(\theta)$$

$$z = \sin(\theta)$$

where x, y and z are as in Figure 3.1. Note that the roll angle ψ turns out to be irrelevant – after all, 'rolling' my head to either side does not change the direction I am looking. The same cosine loss function can be used for this representation as it could in the 2-d vector case. Although this representation may appear to contain less information than that used by Prokudin et. al., since it discards the roll angle, we will see later in our uncertainty model that it actually provides more specific information.

3.2 Von Mises-Fisher Distribution

Analogous to the use of Gaussian distributions as the building blocks of mixture models on the line and the plane, Prokudin et. al.'s paper used von Mises

distributions as the building blocks of their finite and infinite mixture models. Von Mises distributions are close approximations of Gaussians on the circle, with the benefit of being much simpler and more computationally tractable.

A similar distribution on spheres of higher dimension (in our case, S^2, the ordinary sphere), called the von Mises-Fisher (VMF) distribution, will be our building block. It has probability density function

$$p(\boldsymbol{y}; \boldsymbol{\mu}, \kappa) = C_d(\kappa) \exp(\kappa \boldsymbol{\mu}^T \boldsymbol{y})$$

where d is the dimensionality, $\boldsymbol{\mu}$ is the mean vector, κ is a measure of concentration (analogous to $1/\sigma^2$) and $C_d(\kappa)$ is a normalisation constant. In our case $d = 3$ and

$$C_3(\kappa) = \frac{\kappa}{4\pi \sinh \kappa} = \frac{\kappa}{2\pi(e^{\kappa} - e^{-\kappa})}$$

To emphasise, this is the key difference between this paper and that of Prokudin et. al.: whereas they used the von Mises distribution to predict three separate distributions on the circle, one for each of the three angles, we use VMF distributions to predict a *single* distribution for gaze direction on the *sphere*.

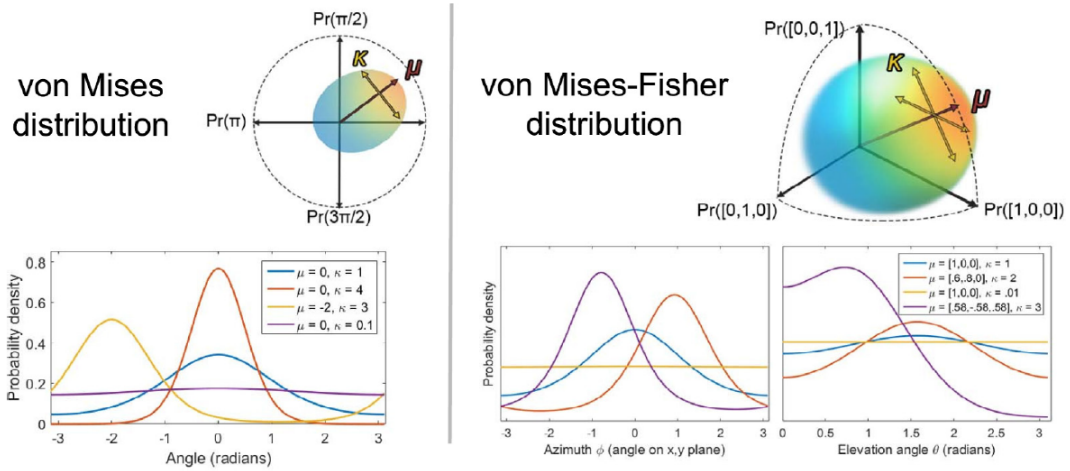


Figure 3.2: Parameters and shapes of von Mises and von Mises-Fisher (3-dimensional case) distributions [Kva19].

Let's start with a single such distribution. Our goal is to predict the parameters of the distribution that will give us the greatest likelihood function for our target vector. To this end, we need four outputs: three of which will be normalized to a unit vector, as before, which will constitute $\boldsymbol{\mu}$ – the centre of

the distribution. The fourth, which must be restricted to be non-negative, is κ, our concentration parameter. The pdf of our predicted distribution is now

$$p_\theta(\mathbf{y}|\mathbf{x}) = \frac{\kappa_\theta(\mathbf{x})}{2\pi\left(e^{\kappa_\theta(\mathbf{x})} - e^{-\kappa_\theta(\mathbf{x})}\right)} \cdot \exp(\kappa_\theta(\mathbf{x})(\boldsymbol{\mu}_\theta(\mathbf{x}))^T\mathbf{y})$$

where $\mathbf{x}$ is an input image, $\mathbf{y}$ is the target vector, θ are the network parameters, and $\boldsymbol{\mu}_\theta(\mathbf{x})$ and $\kappa_\theta(\mathbf{x})$ are the network outputs.

We then train the network by maximising the log-likelihood:

$$\log \mathcal{L}(\theta|\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{N} \kappa_\theta(\mathbf{x}^{(i)})(\boldsymbol{\mu}_\theta(\mathbf{x}^{(i)}))^T\mathbf{y}^{(i)} + \sum_{i=1}^{N}(\log \frac{\kappa_\theta(\mathbf{x}^{(i)})}{2\pi\left(e^{\kappa_\theta(\mathbf{x}^{(i)})} - e^{-\kappa_\theta(\mathbf{x}^{(i)})}\right)})$$

As the network trains, we expect two things to happen: first, the predicted values of $\boldsymbol{\mu}$ to become closer to the target vector (increasing accuracy), and second, the predicted values of κ to be, on average, higher for more accurate predictions than for less accurate predictions, as the network learns to recognise 'difficult' cases – uncertainty.

3.3 Finite Mixture of von Mises-Fisher Distributions

The unimodal model described above is probabilistic, and can account for uncertainty. We can improve on this, however, again taking our lead from Prokudin et. al. Consider a toy case such as the one below: the pose orientation looks to be in one of two directions, though it is hard to say which.

Figure 3.3: An ambiguously oriented face

Our single VMF distribution cannot capture this information – it can only

predict a single centre, and probability density is assumed to decay with distance from this centre uniformly in all directions. The solution, then, is to output not just one distribution, but several, each with a distinct $\boldsymbol{\mu}$, κ and weight, π. These distributions together form our mixture distribution, with probability density

$$p_\theta(\boldsymbol{y}|\boldsymbol{x}) = \sum_{j=1}^{K} \pi_j(\boldsymbol{x}, \theta) p_j(\boldsymbol{y}|\boldsymbol{x}, \theta)$$

where $p_j(\boldsymbol{y}|\boldsymbol{x}, \theta)$ for j= $1, ..., K$ are the K component VMF distributions and $\pi_j(\boldsymbol{x}, \theta)$ are the mixture weights with $\sum_{j=1}^{K} \pi_j(\boldsymbol{x}, \theta) = 1$.

Here K, the number of components in our mixture, is a model hyper-parameter. We require our network now to produce $5K$ outputs: three for the normalized unit vector $\boldsymbol{\mu}$, one for concentration parameter κ, and one weight π, for each of the K component distributions, where the πs are softmaxed to produce a weight vector. As in the single distribution case, we train by maximising the log-likelihood.

We note that Prokudin et. al. introduced this finite mixture of, in their case, von Mises distributions, before going on to extend this method to an infinite mixture of distributions via a sequence of transformations. Their paper does not present a direct comparison of results between the finite and infinite mixtures. In experiments with VMF distributions, we noted that the improvement in performance quickly became negligible as we increased the number of components K - see Figure 4.3. Therefore we made the decision to proceed with a strictly finite mixture, avoiding the additional conceptual difficulty and computational cost associated with the infinite mixture.

3.4 Advantage of Single Vector Approach

At this point, let us return to the earlier point concerning the information contained in the three-angle versus in the single-vector methods. Consider the image above, where two distinct pose orientations look likely. Assume that the networks are trained successfully enough to distinguish both of these two, and assign roughly equal probabilities to the two. The single-vector method we use would then clearly predict two likely 'hot-spots'. Note that the three-angle method, however, would predict two modes for the yaw angle; two modes for the pitch angle; and potentially two modes for the roll angle – since these are

output independently, this yields up to eight likely orientations (or four if we discount the roll angle). This could possibly be rescued by additionally outputting a correlation matrix between components – however, the single-vector method retains the information in a far simpler way.

3.5 Point Prediction

Obtaining a single point prediction from a mixture of distributions necessitates sampling. Let us define the optimal point prediction as the Minimum Bayes Risk (MBR) predictor - that which minimises the expected error:

$$\hat{\boldsymbol{y}} = \arg\min_{\boldsymbol{y'} \in S^2} \sum_{\boldsymbol{y} \in S^2} p(\boldsymbol{y}|\boldsymbol{x}) l(\boldsymbol{y}, \boldsymbol{y'})$$

where S^2 is the 2-sphere (the domain of valid predictions) and $l(\boldsymbol{y}, \boldsymbol{y'})$ is the loss function of interest, in this case the mean absolute angular error (MAAE).

As Premachandran et. al. [PTB14] point out, this is 'doubly intractable': the summation and minimisation are both over an infinite set. We therefore generate a point prediction as follows:

- Sample N points from the mixture distribution
- Calculate the distance matrix of the N points
- Choose the point from the sample with the least mean distance to the other points of the sample.

In theory, for sufficiently large N, this converges to the optimal point. In tests, we tried various values for N and found that $N = 1000$ works as well as any larger value. Note that this MBR point prediction is the same as that used by Prokudin et. al., but our sampling technique is simpler.

Chapter 4

Experiments

4.1 Data and Implementation Details

The dataset used for experiments is Annotated Facial Landmarks in the Wild [KWRB11], a large dataset of some 21,997 images in natural settings (hence 'in the wild'), from which we were able to extract 24,984 usable cropped images of faces. Among its useful annotations are the Euler angles we require.

Data was split into three sets in the ratio 70/20/10. These form the training, validation and final tests sets. The majority of evaluation is on the validation set – the final test set is only used after the best network has been selected, to produce an unbiased performance metric for comparison with benchmarks.

Implementation is in Tensorflow, with the majority of network details implemented using the Keras front end. Optimisation was done using the Adam optimiser to minimise the negative log-likelihood of the data. The number of training epochs was 100. A batch size of 32 images was used, except for the Inception-v3 with input resolution 149, for which this was reduced to 16 due to training memory constraints. Source code is available at https://github.com/RayKMAllen/Headpose.

4.2 Augmentation

In training it was noticed that the probabilistic model overfits more easily than an equivalent simple point estimation model (Figure 4.2). To address this, we employ a number of data augmentation techniques: shifting the image horizontally and/or vertically by a random distance (random up to 0.2 times

(a) No augmentation. (b) Shift (c) Flip (d) Rotate (e) Cutout (f) All

Figure 4.1: Augmentation techniques used.

the size of the image), randomly flipping the image horizontally and/or vertically (with probability 0.5 for each axis), rotating the image (random up to 90 degrees in either direction), and applying Cutout [DT17] (probability 0.5, erased proportion random up to 0.4). Applying these on the fly during training entails a time overhead, therefore we sought to apply as few as was necessary to avoid overfitting – however, experiments showed that for the larger networks, having the most parameters, all four techniques were needed.

L_2 regularisation (constant 0.01) was also applied to convolutional and dense layers during training as an additional regulariser.

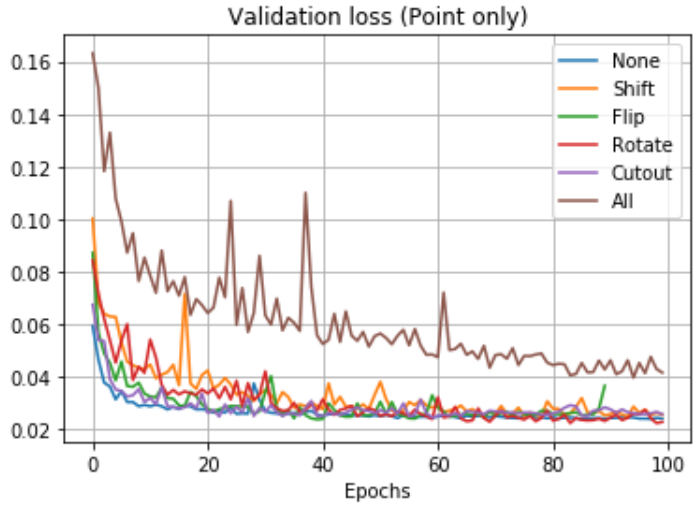
4.3 Number of Mixture Components

As mentioned in section 3.3, K, the number of components of our mixed VMF distribution, is a network hyper-parameter. To determine which value of K to use for later experiments, a VGG was trained for 100 epochs at a selection of trialled values (Figure 4.3). It can be seen that improvements quickly dwindle to insignificance for $K > 2$. This makes some intuitive sense – on looking at an image, we might identify more than one 'most likely' orientation, but it is hard to imagine identifying more than two or three. Further components may help slightly to improve the shape of non-symmetric distribution(s) at the primary node(s).

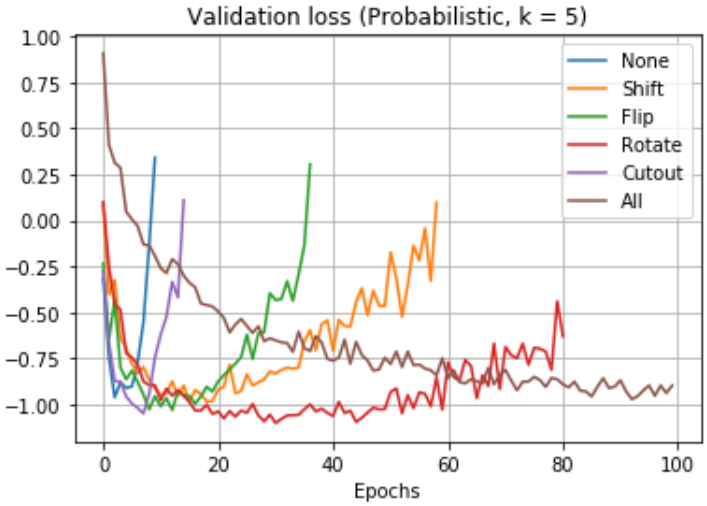
We choose a value of $K = 10$ for subsequent experiments as being large enough without becoming overly cumbersome.

4.4 Networks and Resolutions

We use two metrics for our final evaluation and comparison. Our primary metric is log-likelihood, a standard metric for assessing the quality of probabilistic predictions. We also evaluate MAAE, in order to facilitate comparison to non-probabilistic benchmarks, and also as a more intuitively



(a) Point estimation only.



(b) Probabilistic model.

Figure 4.2: Validation set loss using different augmentation techniques (VGG network).

understandable measure of accuracy.

The training time, prediction time for one image, and number of parameters of each network at each input resolution are also shown for comparison (Tables 4.1, 4.2, 4.3). Tables 4.4 and 4.5 summarise the log-likelihood and MAAE across all architectures and resolutions.

Finally we select the best performing network based on these results - the Inception-v3 with input resolution 149 pixels, compute the final test set MAAE using this trained network, and compare with some benchmark results on the same dataset (Table 4.6). These papers [PC17, Rie18] are suitable for

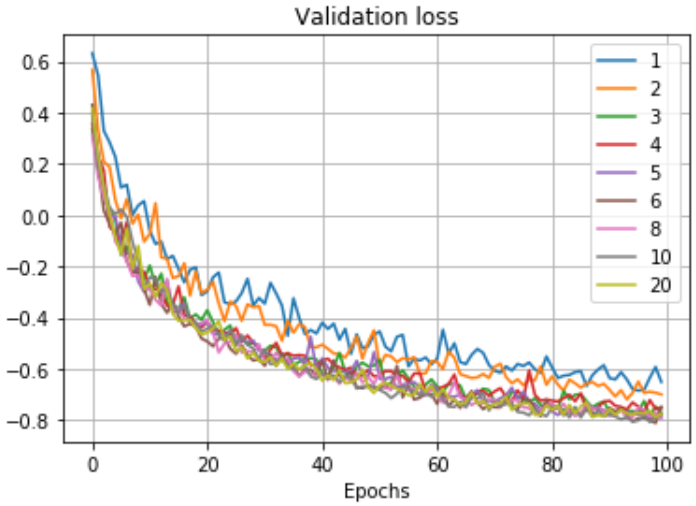


Figure 4.3: Validation set loss by number of mixture components (VGG network).

Table 4.1: Results for input resolution 32x32 pixels.

Network	Log-Likelihood	MAAE	Train time (hrs)	Pred. time (s)	Params (x 10^6)
LeNet-5	0.2653	16.4213°	0.7677	0.004406	1.14
VGG-16	1.0114	10.8121°	3.4109	0.007676	23.18
ResNet-34	0.9853	11.1707°	3.2011	0.010898	20.33
GoogLeNet	0.831	10.5678°	2.9118	0.010299	6.04
Inception-v3	1.2101	9.7338°	13.3306	0.022259	21.91

Table 4.2: Results for input resolution 64x64 pixels.

Network	Log-Likelihood	MAAE	Train time (hrs)	Pred. time (s)	Params (x 10^6)
LeNet-5	0.4612	14.5027°	1.8911	0.005983	4.29
VGG-16	1.1445	10.0103°	5.9894	0.010905	23.18
ResNet-34	1.1109	10.223°	5.4367	0.012606	21.32
GoogLeNet	1.0157	10.0363°	4.1163	0.011103	6.04
Inception-v3	1.2895	9.2913°	16.5405	0.026805	21.91

Table 4.3: Results for input resolution 112x112 pixels (149x149 pixels for Inception-v3).

Network	Log-Likelihood	MAAE	Train time (hrs)	Pred. time (s)	Params (x 10^6)
LeNet-5	0.4694	14.5341°	4.4652	0.013531	12.94
VGG-16	1.1668	9.9325°	9.2477	0.013512	21.22
ResNet-34	1.1424	10.1042°	7.6285	0.014932	21.32
GoogLeNet	1.0304	9.9437°	6.1322	0.014547	6.04
Inception-v3	1.3005	9.3134°	17.4486	0.026951	21.91

comparison as they used similar CNN architectures but were not probabilistic. Since their predictions are for yaw, pitch and roll separately, we convert these to MAAE for comparison using the formula $MAAE = \arccos(\cos\phi \cdot \cos\theta)$, where ϕ and θ are the yaw and pitch angles, as before.

Table 4.4: Summary of log-likelihood results across all networks and input resolutions.

Input resolution	LeNet-5	VGG-16	ResNet-34	GoogLeNet	Inception-v3
32	0.2653	1.0114	0.9853	0.831	1.2101
64	0.4612	1.1445	1.1109	1.0157	1.2895
112/149	0.4694	1.1668	1.1424	1.0304	**1.3005**

Table 4.5: Summary of MAAE results across all networks and input resolutions.

Input resolution	LeNet-5	VGG-16	ResNet-34	GoogLeNet	Inception-v3
32	16.4213°	10.8121°	11.1707°	10.5678°	9.7338°
64	14.5027°	10.0103°	10.223°	10.0363°	9.2913°
112/149	14.5341°	9.9325°	10.1042°	9.9437°	**9.3134°**

Table 4.6: Comparison of test set results with those of benchmarks on the AFLW dataset.

Author(s)	Yaw	Pitch	MAAE
Patacchiola & Cangelosi	9.51°	6.8°	11.67°
Rieger	9.32°	6.18°	11.17°
Ours	n/a	n/a	**9.19°**

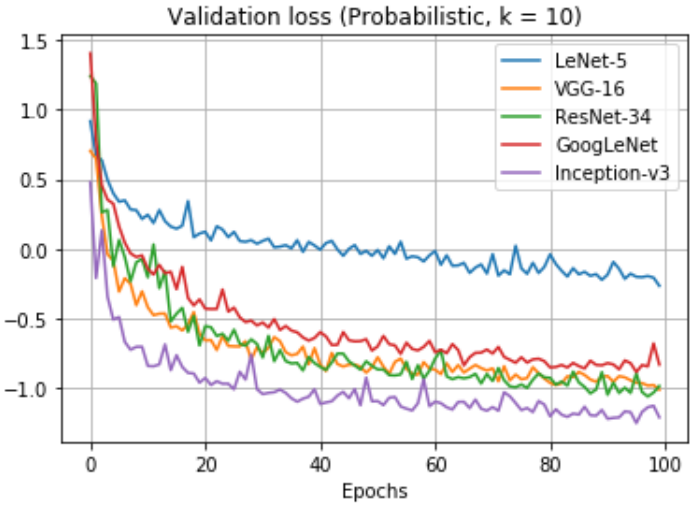
4.5 Discussion of Results

The Inception-v3 network achieves the best results at all resolutions in terms of log-likelihood and MAAE, at the cost of a much longer training time. VGG-16, ResNet-34 and GoogLeNet achieve roughly similar results, while as expected the LeNet-5 lags a long way behind, lacking enough power to produce good results.

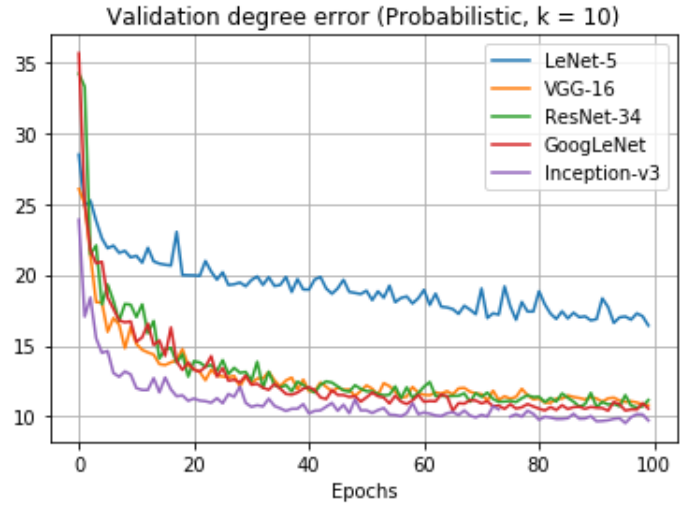
Impressively, the point predictions of our best model are better than those of benchmarks - in fact, among all our network architectures except the LeNet-5, and all 3 input resolutions, only the ResNet-34 at 32x32 pixels fails to outperform benchmarks. We are able to consistently produce a MAAE of approximately 10°. As a caveat, it should be noted that the benchmark results are cross-validated, unlike ours - training time made this unachievable during the time available for this paper.

The prediction time of the Inception-v3 is roughly twice that of the other networks other than the LeNet-5. Whether this increased latency is of practical significance is a question for those applying such networks in a practical context - such as autonomous driving control - to consider.

Also of such practical interest is the parameter count of the four best networks - at 6 million parameters, the GoogLeNet is much lighter on memory than the other three, which all have a little over 20 million. Interestingly, the



(a) Loss (negative log-likelihood)

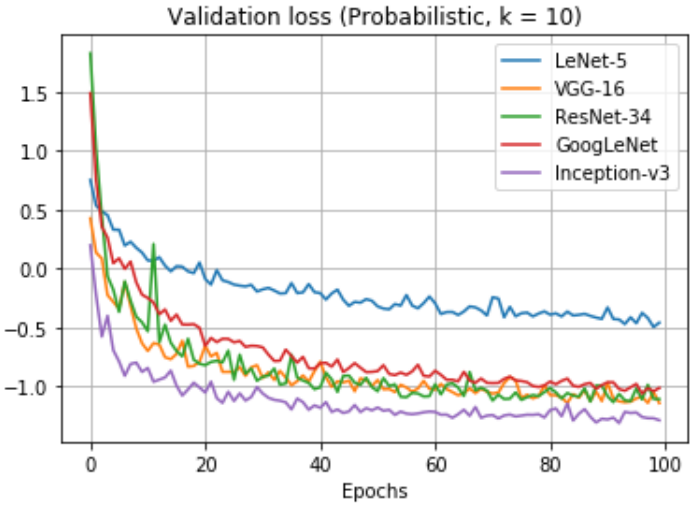


(b) MAAE

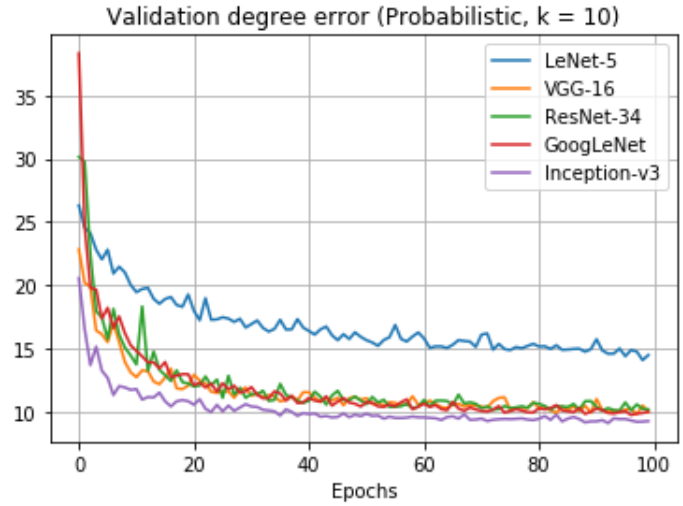
Figure 4.4: Validation set performance by network, input resolution 32x32 pixels.

Inception-v3, once trained, is as lightweight as the VGG or ResNet.

In terms of input resolution, performance at size 64 and 112/149 are on a par, with 32 lagging behind. Again, the practical implications of this are application-dependent, with the most likely significant conclusion being the lack of improvement associated with the step up from 64x64 pixels.

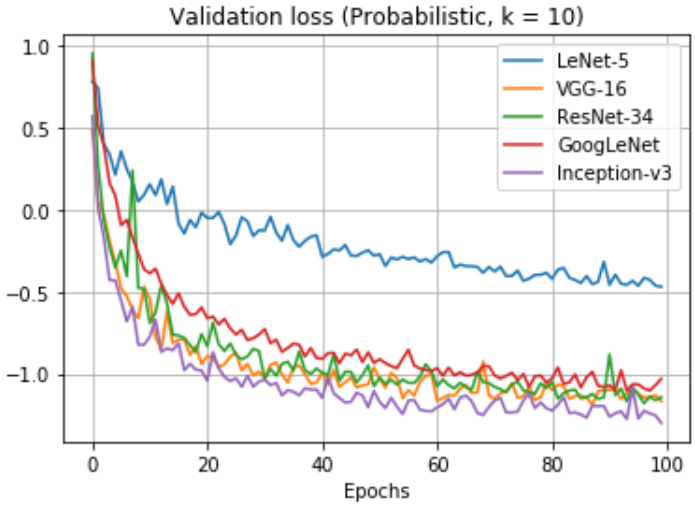


(a) Loss (negative log-likelihood)

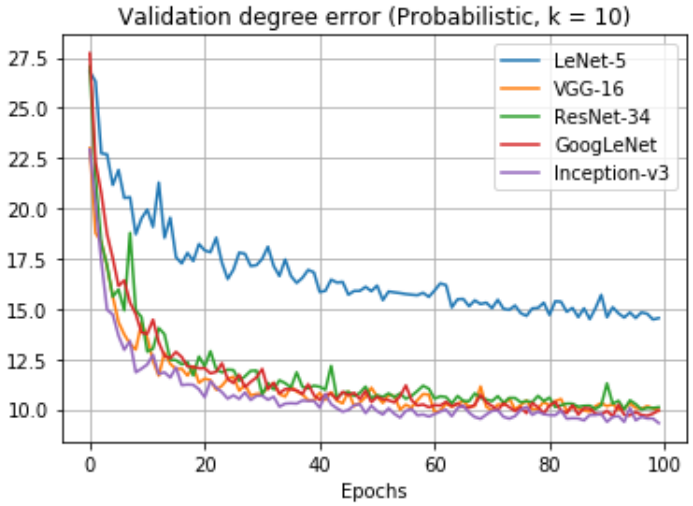


(b) MAAE

Figure 4.5: Validation set performance by network, input resolution 64x64 pixels.

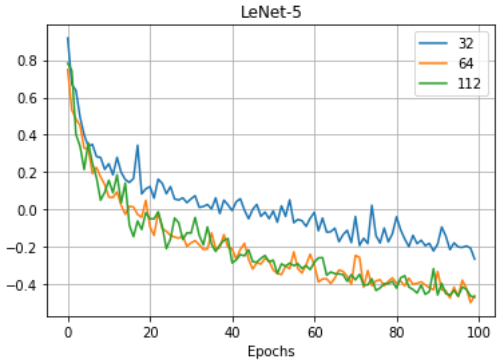


(a) Loss (negative log-likelihood)

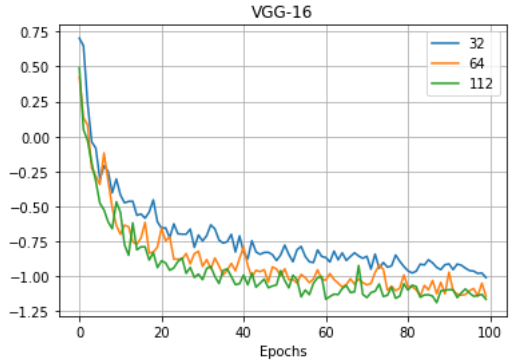


(b) MAAE

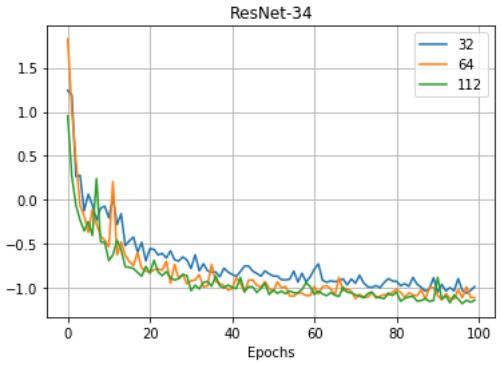
Figure 4.6: Validation set performance by network, input resolution 112x112 pixels (149x149 pixels for Inception-v3).



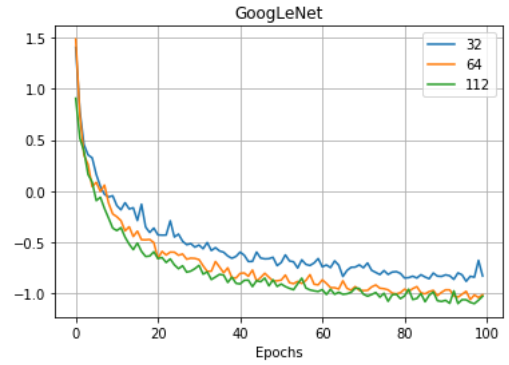
(a) LeNet-5



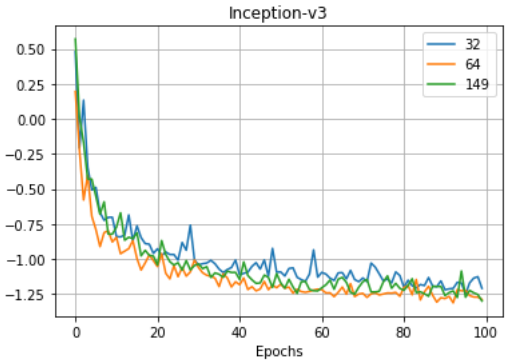
(b) VGG-16



(c) ResNet-34



(d) GoogLeNet



(e) Inception-v3

Figure 4.7: Validation set loss (negative log-likelihood) chart by input resolution for each network.

Chapter 5

Conclusion and Future Work

We have demonstrated a model for head pose orientation estimation which is probabilistic in nature, and as such has the attractive feature of producing an estimate of its own confidence or uncertainty in its predictions. By using von Mises-Fisher distributions on S^2, we have reduced previous models which predicted 3 angles separately to a single prediction, improving the specificity of predictions over such models.

Experiments indicate that point prediction accuracy results are improved on those of other work utilising the same dataset. Having evaluated results using a range of CNN network bodies and input resolutions, we find that inputs of 64x64 pixels are as effective as larger images. Of the CNN models implemented, the Inception-v3 produced the best results in terms of log-likelihood and mean absolute angular error, with the latter consistently coming in at less than 10°.

It would be desirable to directly compare results between our VMF-based model and the simple von Mises-based, 3-angle model of Prokudin et. al., by applying both to the same dataset. Comparisons across different datasets have little value.

We have made the observation that a network which outputs a measure of its confidence in its own predictions is useful - however we have not attempted to articulate how exactly uncertainty measures should be incorporated into applications. In an autonomous driving context, one approach might be to derive a high confidence interval from the generated mixture distribution, and use this as a safety check for the driver control system. Alternatively, if the control system is itself a neural network, the uncertainty (in the form of log-likelihood of the point prediction, or another form) would constitue an input

parameter. We leave this to the designers of such applications.

Input data in this paper was in the form of still images - future work may apply the same framework to RGB-D [GZC+18] or video data.

In terms of efficient use of CPU space and optimal input resolution, other network architectures than those evaluated here may further improve performance - in particular, it would be interesting to see how effective a neural architecture search method such as Efficient Neural Architecture Search (ENAS [PGZ+18]), optimised directly on the task of head pose estimation, would prove, as its block-cell structure can be specifically tailored to the constraints of the system.

References

[APK14] Byungtae Ahn, Jaesik Park, and In So Kweon. Real-time head orientation from a monocular camera using deep neural network. In *Asian conference on computer vision*, pages 82–96. Springer, 2014.

[DT17] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[EG10] Markus Enzweiler and Dariu M Gavrila. Integrated pedestrian classification and orientation estimation. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 982–989. IEEE, 2010.

[FDGKG14] Fabian Flohr, Madalin Dumitru-Guzu, Julian FP Kooij, and Dariu M Gavrila. Joint probabilistic pedestrian head and body orientation estimation. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 617–622. IEEE, 2014.

[FGVG11] Gabriele Fanelli, Juergen Gall, and Luc Van Gool. Real time head pose estimation with random regression forests. In *CVPR 2011*, pages 617–624. IEEE, 2011.

[FY11] Ryusuke Furuhashi and Keiichi Yamada. Estimation of street crossing intention from a pedestrian's posture on a sidewalk using multiple image frames. In *The First Asian Conference on Pattern Recognition*, pages 17–21. IEEE, 2011.

[GX14] Xin Geng and Yu Xia. Head pose estimation based on multivariate label distribution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1837–1842, 2014.

[GZC+18] Yudong Guo, Juyong Zhang, Lin Cai, Jianfei Cai, and Jianmin Zheng. Self-supervised cnn for unconstrained 3d facial performance capture from a single rgb-d camera. *arXiv preprint arXiv:1808.05323*, 2018.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[Kva19] Peter D Kvam. A geometric framework for modeling dynamic decisions among arbitrarily many alternatives. *Journal of Mathematical Psychology*, 91:14–37, 2019.

[KWRB11] Martin Koestinger, Paul Wohlhart, Peter M Roth, and Horst Bischof. Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization. In *2011 IEEE international conference on computer vision workshops (ICCV workshops)*, pages 2144–2151. IEEE, 2011.

[LGS19] Shuai Liao, Efstratios Gavves, and Cees GM Snoek. Spherical regression: Learning viewpoints, surface normals and 3d rotations on n-spheres. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9759–9767, 2019.

[LJB+95] Yann LeCun, LD Jackel, Leon Bottou, A Brunot, Corinna Cortes, JS Denker, Harris Drucker, I Guyon, UA Muller, Eduard Sackinger, et al. Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60, pages 53–60. Perth, Australia, 1995.

[LT12] Jiwen Lu and Yap-Peng Tan. Ordinary preserving manifold analysis for human age and head pose estimation. *IEEE Transactions on Human-Machine Systems*, 43(2):249–258, 2012.

[MG18] Agnieszka Mikołajczyk and Michał Grochowski. Data augmentation for improving deep learning in image classification

problem. In *2018 international interdisciplinary PhD workshop (IIPhDW)*, pages 117–122. IEEE, 2018.

[PC17] Massimiliano Patacchiola and Angelo Cangelosi. Head pose estimation in the wild using convolutional neural networks and adaptive gradient methods. *Pattern Recognition*, 71:132–143, 2017.

[PGN18] Sergey Prokudin, Peter Gehler, and Sebastian Nowozin. Deep directional statistics: Pose estimation with uncertainty quantification. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 534–551, 2018.

[PGZ+18] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[PTB14] Vittal Premachandran, Daniel Tarlow, and Dhruv Batra. Empirical minimum bayes risk prediction: How to extract an extra few% performance from vision models with just three more parameters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1043–1050, 2014.

[Rie18] Ines Sophia Rieger. *Head Pose Estimation using Deep Learning*. PhD thesis, Ph. D. thesis, University of Bamberg, 2018.

[SIVA17] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[SM15] G Santoshi and SR Mishra. Pedestrian with direction detection using the combination of decision tree learning and svm. In *Emerging ICT for Bridging the Future-Proceedings of the 49th Annual Convention of the Computer Society of India (CSI) Volume 1*, pages 249–255. Springer, 2015.

[SQLG15] Hao Su, Charles R Qi, Yangyan Li, and Leonidas J Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2686–2694, 2015.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[TK14] Junli Tao and Reinhard Klette. Part-based rdf for direction classification of pedestrians, and a benchmark. In *Asian Conference on Computer Vision*, pages 418–432. Springer, 2014.

[TM15] Shubham Tulsiani and Jitendra Malik. Viewpoints and keypoints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1510–1519, 2015.

[VDMK15] DK Vishwakarma, Ashish Dhiman, Rockey Maheshwari, and Rajiv Kapoor. Human motion analysis by fusion of silhouette orientation and shape features. *Procedia Computer Science*, 57:438–447, 2015.

[XWCL15] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

Appendix A

Network Architectures

Chapter 3 describes the network 'head' – the last few layers of the network, which specify the format we wish our outputs to take. The 'body' of the network can be any neural network capable of learning from image data - CNN variants being the obvious choice. Here we briefly describe the networks used to evaluate our probabilistic model.

Since one of our aims is to investigate the effectiveness of different input image resolutions, we also describe adjustments made to each network in the case of non-standard resolutions. In general, there is no standard method for adjusting CNNs in this way; we have taken the general approach of maintaining each network's depth as canonically described, adjusting only dimension reduction layers such as max-pooling. By doing so we have hoped to keep the internal structure and functionality of each network as constant as possible across different input dimensionalities.

A.1 Note on Activation Functions

The standard activation function for the following models is the rectified linear (ReLU) function. This paper uses the leaky ReLU function instead, as the leaky version has been shown empirically to outperform ordinary ReLU [XWCL15]. For the sake of avoiding excessively cluttered illustrations, we have omitted the activations from diagrams.

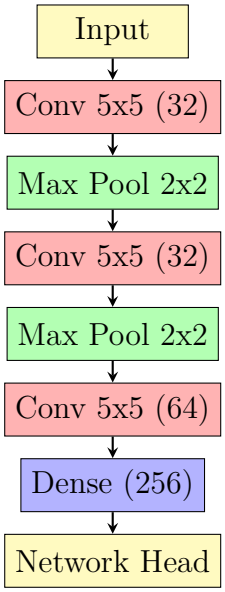


Figure A.1: LeNet-5

A.2 LeNet-5

This is among the simplest commonly used convolutional architectures. It consists of three 5x5 convolutional layers with ReLU activations, the first two of which are followed by max-pooling layers. The final activated convolutional layer is followed by a dense layer before being passed to the network head.

For this network, we make no adjustments based on image resolution.

A.3 VGG-16

Another conceptually simple CNN, the VGG amounts basically to an expanded LeNet, consisting of sequential convolutional layers, max-pooling layers for dimension reduction, and two dense layers.

The default image resolution of this network is 224x224 pixels. We tweak this to handle images of size 112, 64 or 32 by omitting max-pooling layers as early as possible in the network.

A.4 ResNet-34

This network is based on the concept of residual blocks featuring 'skip' connections (see original paper [HZRS16] for a full description). These are believed to reduce the 'vanishing gradient' problem. Batch normalisation plays

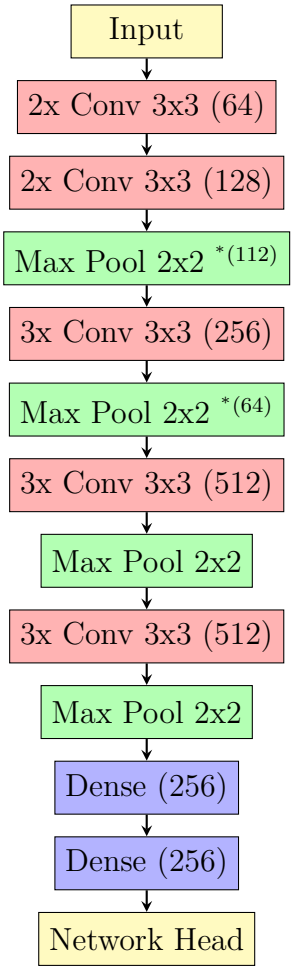


Figure A.2: VGG-16. $^{*(n)}$: Layer included only if input resolution $\geq n$.

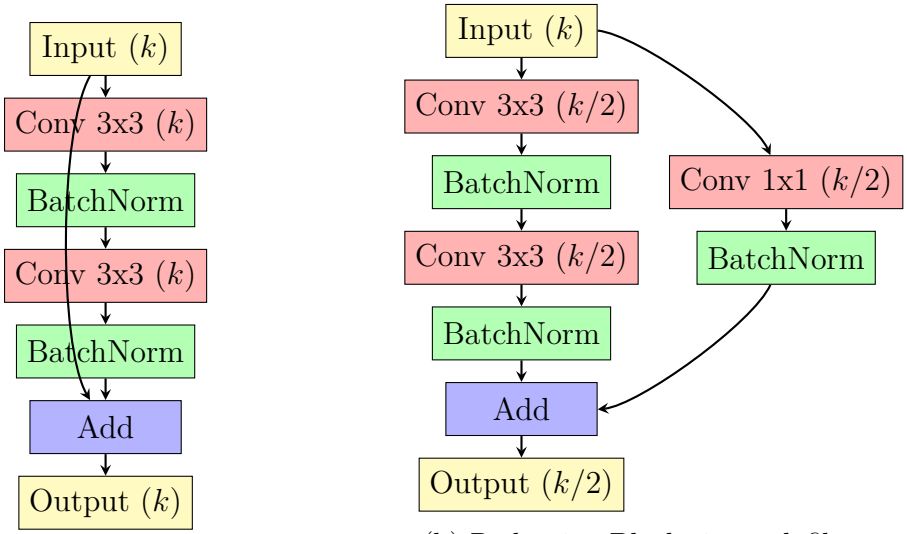
an important role.

ResNet-34 consists as shown of a 7x7 convolutional layer, followed by a sequence of stacked residual blocks, each of which either keeps the output dimension constant (identity block) or halves it (reduction block). Rather than passing through a dense layer before being passed to the network head, a global average pooling layer is used in order to reduce the network's number of parameters.

Like the VGG, the ResNet's default input size is 224. We adjust to sizes of 112, 64 or 32 by increasing the stride of the initial convolutional layers to 2, and substituting identity blocks for reduction blocks.

A.5 GoogLeNet

The first 'Inception' network. The basic building block is the inception module, where the input is passed through convolutions of different sizes, as well as a



(a) Identity Block with k filters.

(b) Reduction Block: input k filters, output $k/2$ filters.

Figure A.3: Residual blocks as featured in ResNet-34.

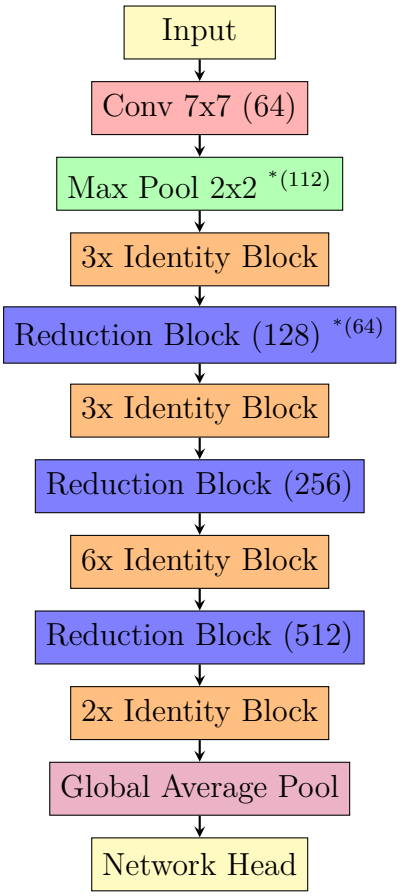


Figure A.4: ResNet-34. $^{*(n)}$: Layer included only if input resolution $\geq n$ for pooling layer, replaced by identity block if reduction block.

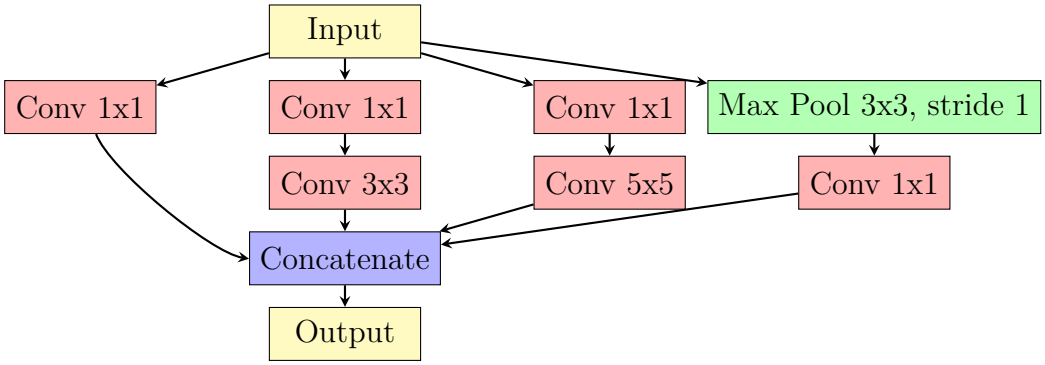


Figure A.5: Inception Module as used in GoogLeNet. Filter numbers vary per individual module.

pooling layer, the outputs of which are then concatenated (see original paper [SLJ+15] for details). The idea is for the network to be able to recognise objects of different sizes in the image.

GoogLeNet consists of a 'stem' of convolution and max-pooling layers, followed by a sequence of inception modules occasionally interspliced with further max-pooling layers. Similarly to the ResNet, a global average pooling layer is used, this time with a dropout layer for regularisation, before being passed to the network head. A weighted 'auxiliary' network head is also used to address the vanishing gradient problem.

Its default input size is also 224. Other input sizes are handled in the same way as for ResNet-34.

A.6 Inception-v3

An elaboration on the GoogLeNet, this network features several different variants on the inception module, making use of factorised convolutions and batch normalisation.

The overall structure is similar to that of its predecessor, with an expanded stem and dedicated reduction inception modules in place of max-pooling. The original version featured an auxiliary training head similar to those of GoogLeNet - in the presence of batch normalisation, the authors noted its function was reduced to that of regularisation. We have omitted it for simplicity.

The default input size for this network, at 299x299 pixels, is larger than for the other networks we have considered. Striding and/or pooling are used to accommodate input sizes of 149, 64 or 32 as shown. Note also that the

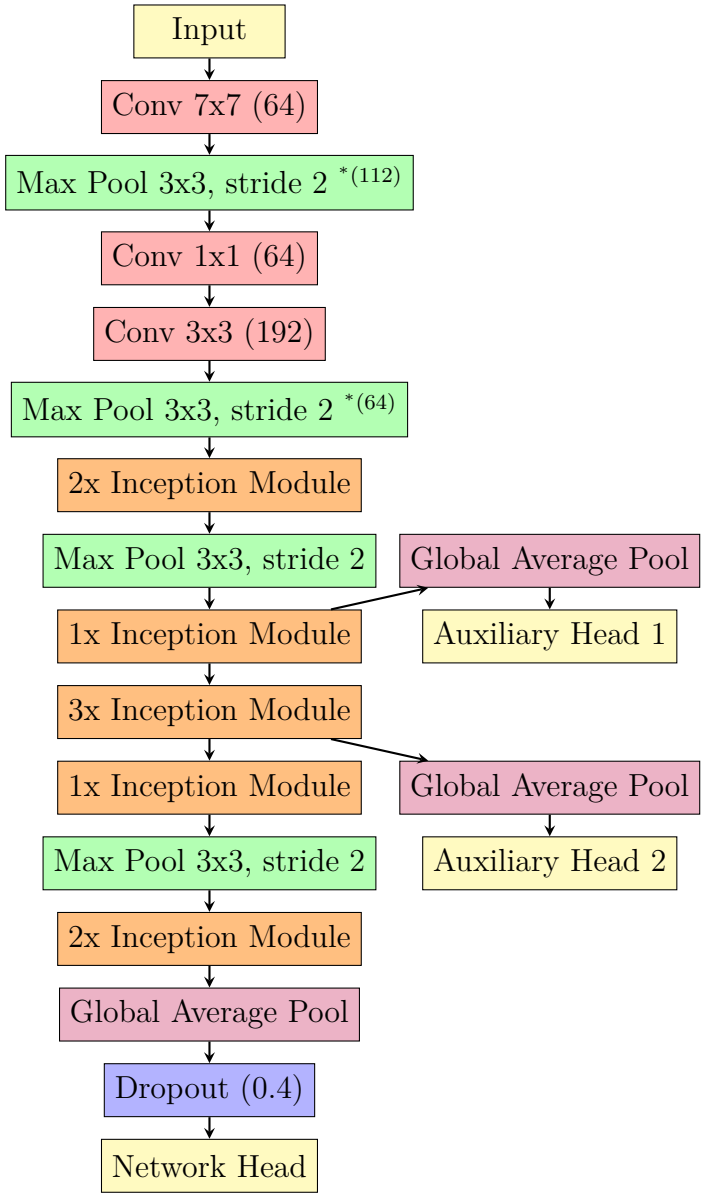


Figure A.6: GoogLeNet. $^{*(n)}$: Layer included only if input resolution $\geq n$.

implementation used here contains fewer inception modules than in the original paper, due to training memory constraints. Inception module C, and inception reduction module B, are shown for illustrative purposes; other modules are conceptually similar (see original paper [SIVA17] for full details.)

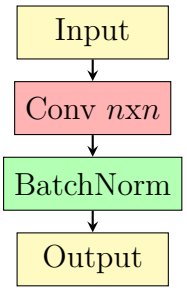


Figure A.7: A convolution block as used in Inception-v3, consisting of a convolution layer (variable filter size) followed by a batch normalization layer.

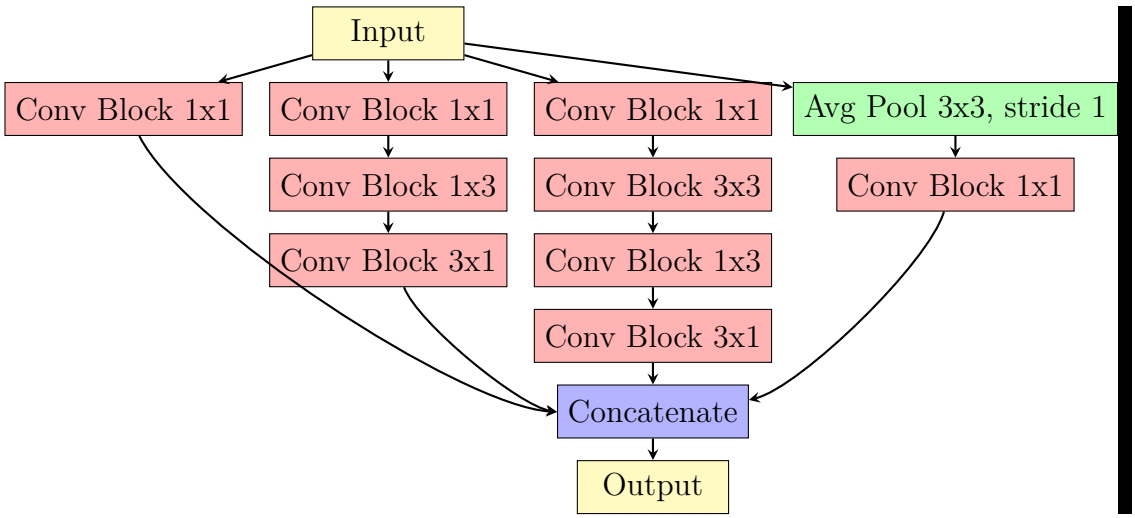


Figure A.8: Inception Module C as used in Inception-v3. Filter numbers vary per individual module.

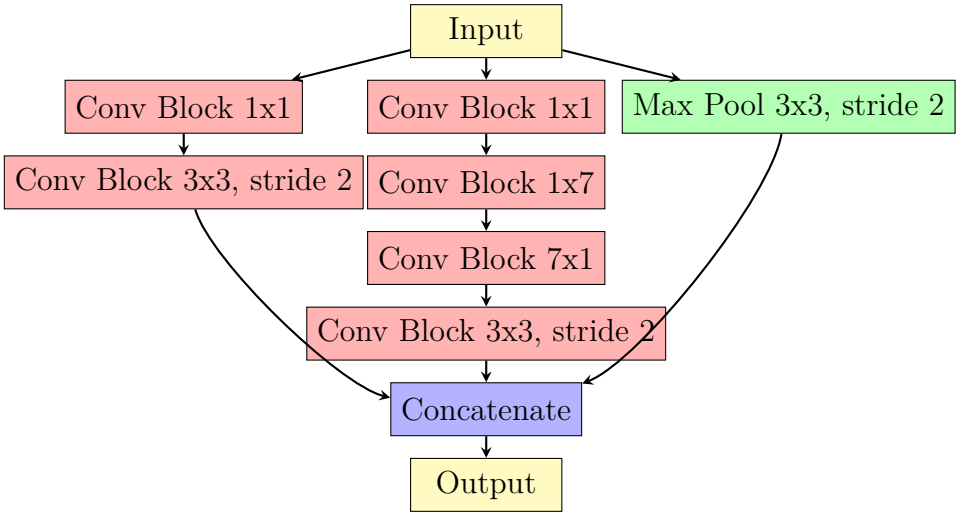


Figure A.9: Inception Reduction Module B as used in Inception-v3.

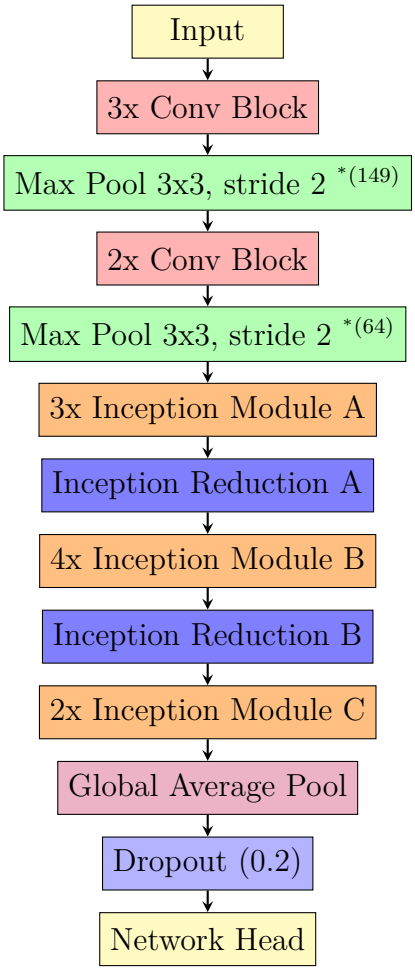


Figure A.10: Inception-v3. $^{*(n)}$: Layer included only if input resolution $\geq n$.

Appendix B

Technical Specifications

Table B.1: Technical Specifications

Name	Information
TensorFlow	1.13.1
Python	3.6.5
Anaconda	4.5.4
CUDA	10.0
cuDNN	7.5.0
GPU	NVIDIA GeForce GTX 1080 (8 GB VRAM)
CPU	Intel Core i7-6800K (1200 CPUs)
Operating System	Windows 10 Enterprise 64-bit (10.0)