Questions: 3.3, 3.5, 4.3, 4.4

Candidate number: H801L

10th December 2020

1 Q 3.3

1.1 Stereo Rectification

Stereo rectification is a process of resampling image planes, of two cameras of differing viewpoints, to reduce the correspondence problem in Stereo Vision from 2D to 1D search, thus simplifying calculations for other stereo algorithms.

We want to identify such image transformations, which will resample two images to form a pair of projections where epipolar lines are aligned between projections and run parallel to x-axis. The images we obtain after resampling lie on the same plane, parallel to the baseline. The epipolar lines are aligned thus disparities are only in the x-direction (1D). Image rectification can be applied to image pairs of both calibrated and uncalibrated cameras [1], [2], [3], [4].

Uncalibrated cameras When cameras are uncalibrated **Hartley's algorithm** is a popular choice for image rectification. It uses Fundamental matrix (found with at least 7 point correspondences), to identify epipoles in both images, and then constructs such Homography matrix H_L which moves the epipole on left image to infinity $\boldsymbol{e} = [a, b, f]^T \longrightarrow \hat{\boldsymbol{e}} = [0, 0, f]^T$. However, H_L has 4 DoF and a poor choice will introduce severe projective distortions. Therefore, we also find a matching Homography matrix H_R to send epipole on right image to infinity, and then try to minimise $\sum_i d\left(\mathrm{H_L x_{L,i}}, \mathrm{H_R x_{L,i}}\right)^2$ to minimise distortions. Finally we resample both images using their corresponding transformation matrices to obtain rectified images. [2]

Calibrated cameras When two cameras are calibrated and their intrinsic (K) and extrinsic parameters (R, T) are known, a much better choice is the **Bouquet's algorithm**. It uses extrinsic parameters to minimise amount of change the reprojection introduces and maximises the common viewing area [3]. It can be summarised in five steps:

1. Identify rotation R_{rect} of right camera which will send the epipole to infinity. $\hat{\boldsymbol{e}} = [0, 0, f]^T$
2. Apply rotation to right image $R_R = R_{rect}$
3. Apply rotation to left image $R_L = R \cdot R_{rect}$
4. Resample pixels for both images $p_L = R_L[x_L, y_L, f_L]^T$, $p_R = R_R[x_R, y_R, f_R]^T$
5. Adjust scale of rectified points $\hat{p} = \frac{f}{z'}[x', y', z']^T$

1.2 Stereo rectification application

Two applications of stereo rectification include constructing Depth map with Block Matching or SDGM algorithms for robotics sensing or **3D reconstruction** for medical imaging. Image rectification could also be used to simplify task of **image stitching** to form panoramic photographs.

1.3 Methodology - Bouquet's algorithm

Assumptions For simplicity of calculations, we assume following simplified camera setup:

- Focal length is the same for both cameras, $f = 1$
- The origin of image plane coincides with principal point, $c_x = 0$ and $c_y = 0$
- Global 3D coordinate system is aligned wiht left camera coordinate system.
- Translation and Rotation matrices of the second camera are known.
- Cameras' optical axis are co-planar and form angle $\alpha = 45°$

$$\mathrm{R} = \begin{bmatrix} \cos(45°) & 0 & -\sin(45°) \\ 0 & 1 & 0 \\ \sin(45°) & 0 & \cos(45°) \end{bmatrix}, \quad \mathbf{T} = \begin{bmatrix} -4\cos(22.5°) \\ 0 \\ 4\sin(22.5°) \end{bmatrix} \tag{1}$$

World setup Figure 1 presents two camera centers and a regular tetrahedron located in 3D space. Their 3D word coordinates are stated in Equation 2.

$$\mathbf{c_L} = \begin{bmatrix} x = 0 \\ y = 0 \\ z = 0 \end{bmatrix}, \quad \mathbf{c_R} = \begin{bmatrix} 3.696 \\ 0 \\ 1.531 \end{bmatrix}, \quad \mathbf{T} = \begin{bmatrix} p_1 = [1, 1, 10] \\ p_2 = [-1, 1, 9] \\ p_3 = [-1, -1, 10] \\ p_4 = [1, -1, 9] \end{bmatrix} \tag{2}$$

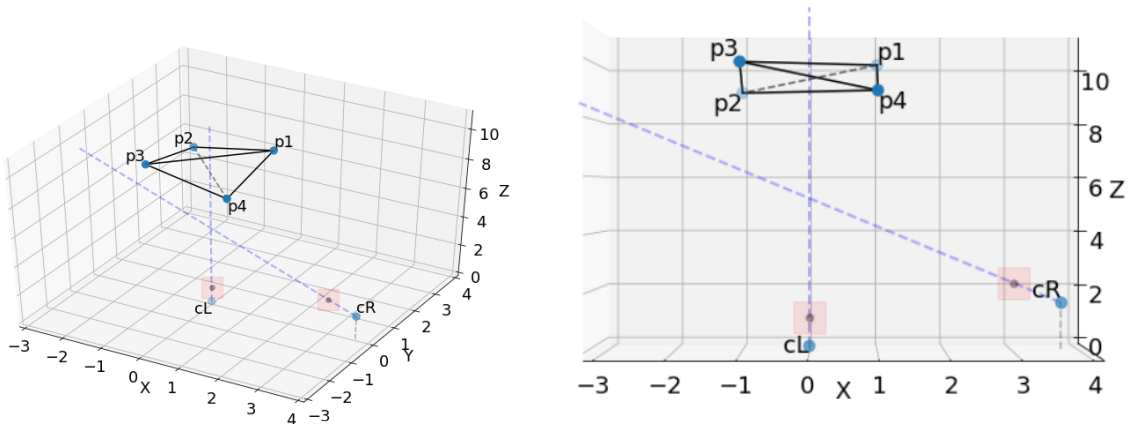
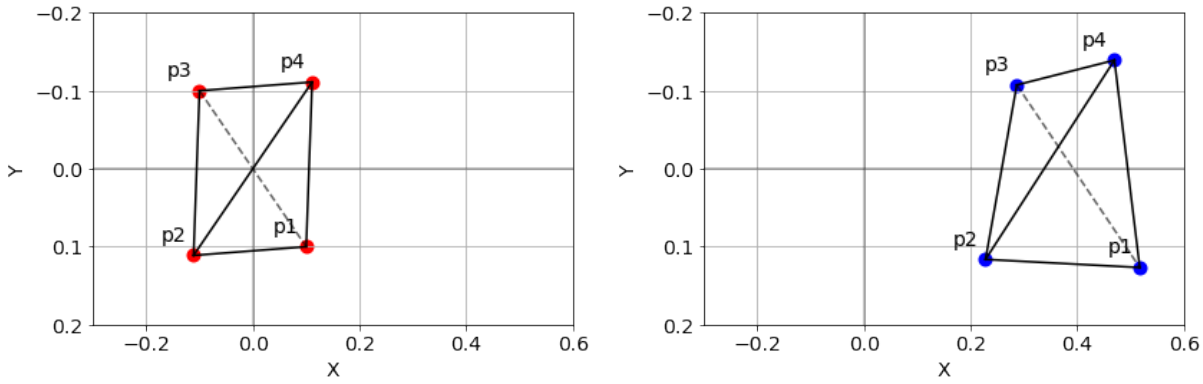


Figure 1: Visualisation of 3D word setup with two co-planar cameras and a tetrahedron.

Figure 2 shows how the tetrahedron projects on the image plane of each camera, with exact 2D projection points listed in Equation 3.



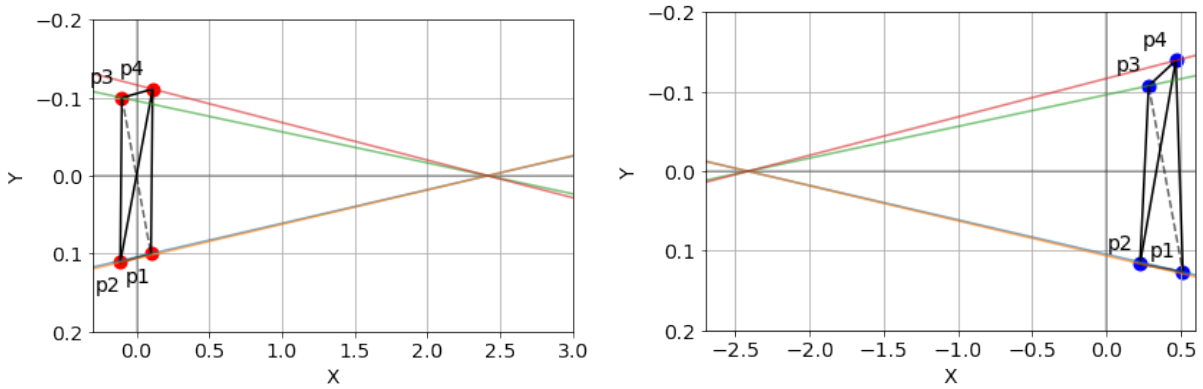
(a) Left image (b) Right image

Figure 2: Projections of the tetrahedron onto image planes of both cameras.

$$\mathbf{T_L} = \begin{bmatrix} p_{L1} = [0.100, 0.100] \\ p_{L2} = [-0.111, 0.111] \\ p_{L3} = [-0.100, -0.100] \\ p_{L4} = [0.111, -0.111] \end{bmatrix}, \quad \mathbf{T_R} = \begin{bmatrix} p_{R1} = [0.517, 0.127] \\ p_{R2} = [0.228, 0.116] \\ p_{R3} = [0.287, -0.107] \\ p_{R4} = [0.470, -0.139] \end{bmatrix} \tag{3}$$

Knowing $\boldsymbol{R}$ and $\boldsymbol{T}$, Essential matrix can be derived in Equation 4 and then used to derive equation for epipolar lines. Figure 3 shows correctly identified epipolar lines.

$$\mathrm{E} = \mathrm{T}_{\times}\mathbf{R}\,, \quad \mathbf{p_R} = \mathrm{E} \cdot \mathbf{p_L}\,, \quad \mathbf{p_R}^T = \mathbf{p_L}^T \cdot \mathrm{E} \tag{4}$$



(a) Left image (b) Right image

Figure 3: Epipolar lines for left and right images. Epipoles are located at $\boldsymbol{e}_L = [2.414, 0, 1]^T$ and $\boldsymbol{e}_R = [-2.414, 0, 1]^T$ respectively.

To rectify images we need to transform them such that epipoles are at infinity, this will be achieved by R_{rect}. In Bouquet's algorithm R_{rect} is found using Equation 5 which is an orthogonal matrix.

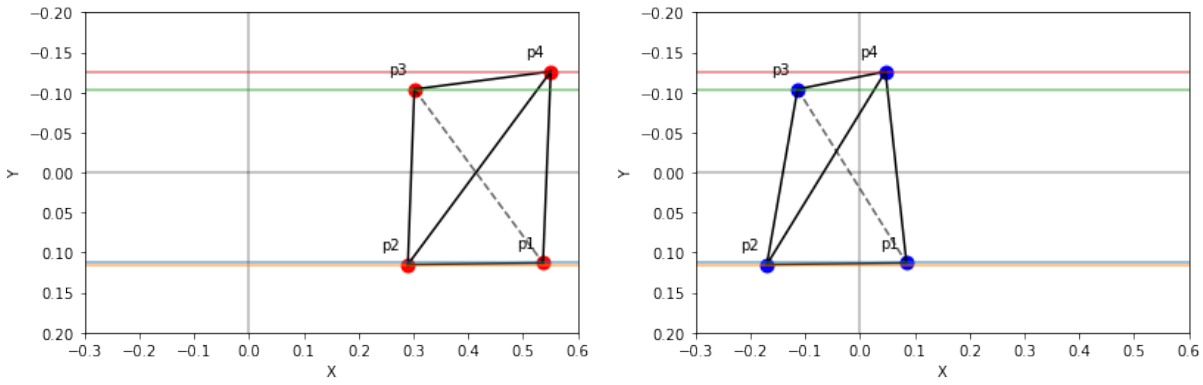
$$\mathbf{R_{rect}} = \begin{bmatrix} \mathbf{e_1}^T \\ \mathbf{e_2}^T \\ \mathbf{e_3}^T \end{bmatrix}, \quad \mathbf{e_1} = \frac{\mathbf{T}}{\| \mathbf{T} \|}, \quad \mathbf{e_2} = \frac{[0,0,1]^T \times \mathbf{e_1}}{\| [0,0,1]^T \times \mathbf{e_1} \|}, \quad \mathbf{e_2} = \frac{\mathbf{e_2} \times \mathbf{e_1}}{\| \mathbf{e_2} \times \mathbf{e_1} \|} \tag{5}$$

Rectified right camera points are given in Equation 7 and 8 alongside with scaled versions. We obtained those with equations:

$$p_R = R_{rect} \cdot [x_R, y_R, f]^T \qquad p_L = R \cdot R_{rect} \cdot [x_L, y_L, f]^T \qquad \hat{p} = \frac{f}{z'}[x', y', z']^T \tag{6}$$

$$\mathbf{T'_R} = \begin{bmatrix} p'_{R1} \\ p'_{R2} \\ p'_{R3} \\ p'_{R4} \end{bmatrix} = \begin{bmatrix} [0.095, 0.127, 1.122] \\ [-0.172, 0.116, 1.011] \\ [-0.118, -0.107, 1.034] \\ [0.051, -0.139, 1.104] \end{bmatrix}, \quad \mathbf{T'_R}^{scaled} = \begin{bmatrix} [0.085, 0.113] \\ [-1.700, 0.115] \\ [-0.114, -0.104] \\ [0.046, -0.126] \end{bmatrix} \tag{7}$$

$$\mathbf{T'_L} = \begin{bmatrix} p'_{L1} \\ p'_{L2} \\ p'_{L3} \\ p'_{L4} \end{bmatrix} = \begin{bmatrix} [0.475, 0.100, 0.886] \\ [0.280, 0.111, 0.966] \\ [0.290, -0.100, 0.962] \\ [0.485, -0.111, 0.881] \end{bmatrix}, \quad \mathbf{T'_L}^{scaled} = \begin{bmatrix} [0.536, 0.113] \\ [0.290, 0.115] \\ [0.302, -0.104] \\ [0.551, -0.126] \end{bmatrix} \tag{8}$$



(a) Left Image (b) Right Image

Figure 4: Rectified images with epipolar lines parallel to x-axis.

Figure 4 shows horizontal epipolar lines for both rectified images. When ploted on a single image plane, Figure 5.a, all corresponding epipolar lines are aligned. Figure 5.b shows how new rectified image plane co responds to the 3D space.

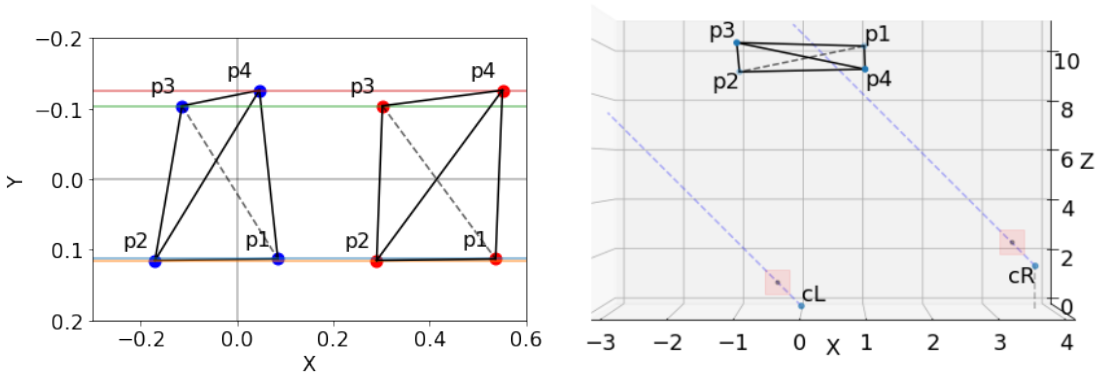


Figure 5: (a) Rectified images overlayed on a single plane. (b) optical axis of both cameras, after rectification, are parallel.

2 Q 3.5

2.1 3D reconstruction algorithms

In this coursework we explore **Space Carving** algorithm by Kutulakos and Seitz [8]. However, before focusing on a single algorithm, we have studied merits and limitations of the following 3D reconstructions approaches [5], [6], [7]

Shape from Silhouettes creates a 3D model called visual hull, at intersections of rays from multiple cameras located around the object. It is computationally fast and independent of surface effects. However, it has limited precision, requires many viewpoints, and can't recover concavities.

Photometric/Carving techniques start with a volume block and itteratively carve away 3D points, for which projections on multiple cameras are not consistent. These are easy to implement and produce complex topologies. However, they often rely on object's texture and are less effective for large scenes or objects which vary in scale.

Disparity based techniques recover 3D structure of a scene by computing disparity map between corresponding points in stereo images to recover scene depth. These methods suffer from correspondence mismatch due to occlusion, reflection, specular highlights, and regions of uniform color and intensity.

Level set methods propose an implicit 3D surface, which is refined to match 3D shape of a scene. Surface is adjusted to improve correlation between its projection onto input images and the image surface. This approach is purely geometric, thus independent of image resolution. However, it cannot model sharp features and the convergence is not guaranteed.

2.2 Space Carving

Space carving algorithm approaches 3D reconstruction as a constraint satisfaction problem. As presented by the flow chart in Figure 6, we start with a 3D Volume containing the entire scene of interest and consisting of a finite number of voxels. Then we itteratively carve the volume into a 3D shape, by removing voxels which projection onto cameras is inconsistent.

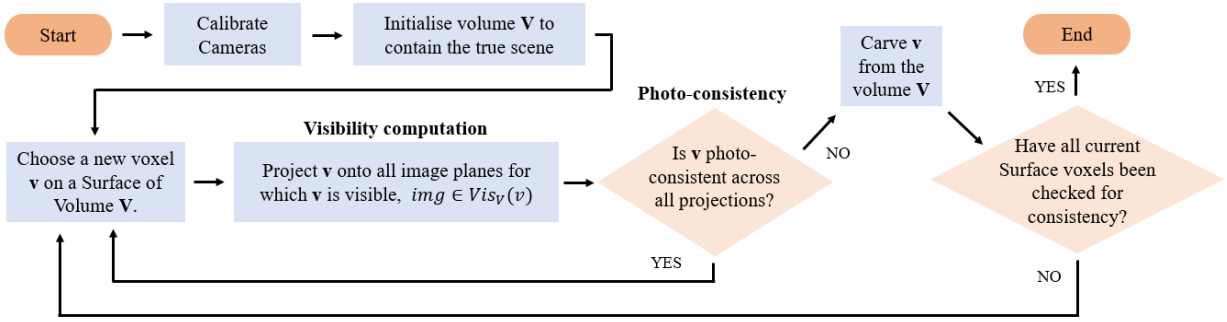


Figure 6: A Flow chart for Space Carving algorithm.

Determining which voxels are visible by each of the cameras (**Visibility computation**) as well as determining whether voxel belongs to a shape or not (**Photo-consistency**) are the two most important pieces of Space carving algorithm which we explain in more detail.

Photo-consistency Space carving algorithm assumes space radiance for which: shadows, transparency, and inter-reflections can be ignored (Lambertian surfaces). Under this assumption, the computation of color, a 3D point would project onto an image, is tractable. Therefore to determine whether a 3D point belongs to a shape or not, we need to check whether its color is consistent for all cameras for which this point is visible. While in theory a single non-photo-consistent projection is sufficient proof to reject a voxel, in practice, to account for errors during image formation, we carve a voxel only if the standard deviation of colors σ is greater than treshold λ, computed in Equation 9. Where K is number of images and σ_0 standard deviation of sensor error.

$$\lambda_v = \frac{(K-1)\sigma^2}{\sigma_0^2} \tag{9}$$

Visibility computation Knowing a 3D location of a voxel and location of all N-cameras we can project the voxel onto cameras and check color consistency. However, how do we determine whether a 3D voxel actually projects onto a camera rather than being occluded by other parts of the 3D shape? Space carving algorithm approaches this problem with **Multi-sweep Space Carving**. We sweep a volume with a plane and test photo-consistency only for those cameras which lie to one side of the plane. This way we maintain order of visibility - occluding voxels are considered before occluded ones. To incorporate all input views we consider several directions of sweeping planes and carve voxels with each one, in turn.

Assumptions and Constraints The main limitation of the Shape Carving algorithm is the simplification of space reflectance (Lambertian surfaces). This assumption will fail when considering transparent or mirror like surfaces. This would also be a problem in a setup with a single camera and a rotating object. In such scenario shadows and illumination changes could produce 3D objects with holes.

Space carving algorithm produces a largest possible photo-consistent reconstruction, therefore the more camera viewpoints we consider, or introduce other known scene constraints, the further we can tighten the bounds on a scene structure and thus obtain a finer 3D model. Use of sufficient number of constraints/viewpoints is especially important when dealing with low-contrast surfaces, such as, walls in large scene reconstruction. There, walls will tend to "buldge" out, because voxel photo-consistency will be ambiguous.

The algorithm focuses on scene recognition under general conditions therefore has no constraints on scene geometry, topology, or camera configuration (although cameras need to be calibrated).

Photo-consistency is considered at a pixel level, thus the algorithm is not robust to pixel noise. This was later addressed by the author [9], by extending photo-consistency evaluation from pixel to region level.

3 Q4.3

3.1 Number of Parameters

Equations used to calculate parameters in each layer are listed below.
Where: p - number of parameters for a layer,

- **Dense-to-Dense**: $p = (n + 1) \times m$
  Where: n - number of inputs, m - number of outputs
- **Conv-to-Conv**: $p = (k \times k \times l + 1) \times r$
  Where: k - filter size, l - input depth, r - number of filters

Parameters in a 2-Layer fully connected network are calculated using Equation 10 and in a 4-Layer convolutional layer using Equation 11.

$$P_{FC} = [(28 * 28 + 1) * 512] \ + \ [(512 + 1) * 10] = 401920 + 5130 = 407050 \tag{10}$$

$$\begin{aligned} P_{CNN} &= [(3 * 3 * 1 + 1) * 32] \ + \ [(3 * 3 * 32 + 1) * 32] \\ &+ [(288 + 1) * 128] \ + \ [(128 + 1) * 10] = \\ &= 320 + 9248 + 36992 + 1290 = 47850 \end{aligned} \tag{11}$$

	fully connected	CNN
number of parameters	407,050	47,850

Table 1: Number of parameters for 2 layer fully connected network and 4 layer CNN.

On parameters Importance The more parameters the network has, the more DRAM memory we need to allocate during training and inference. Training requires more DRAM memory (it has to store gradients in back-propagation), however test time memory is especially important when deploying model onto a portable machine. Additionally, the more parameters in the network, the higher the chance for over-fitting. Therefore for two DNN models with the same model performance we prefer the one with fewer model parameters. Here, 2-Layer Fully connected network has worse performance than CNN (96% vs 97%) while having over 8 times more parameters, which makes it inferior.

However, when using NN for complex task, large number of parameters is necessary in order to implement very deep NNs which are able to learn complex features required to solve such problems. For example, VGG-16 has 138 million parameters and ResNet-152 has 60 million parameters. More recently a DNN in Natural Language Processing, GPT-3, used as many as 185 billion parameters.

Objective function Equation 12 is cross-entropy objective function, where: N - number of training examples, C - number of classification categories (for digit-MNIST $C = 10$), Y - is truth label $Y \in \{0, 1, ...9\}$, S_i is the output of the softmax layer (our prediction). Softmax is defined in Equation 13.

$$L(Y, S) = -\sum_{n=1}^{N}\sum_{i=1}^{C} y_{n,i} log(S_{n,i}) \tag{12}$$

$$S_{n,i} = \frac{e^{a_{n,i}}}{\sum_{j=1}^{C} e^{a_{n,j}}} \tag{13}$$

3.2 Gradient derivation - Dense-10 layer

For simplicity we compute gradient for a single weight and a single input n. Gradients for other weights in a layer can be found by analogy.

$$\frac{\partial L}{\partial w_{d,i}^{(2)}} = \frac{\partial L}{\partial \boldsymbol{S}} \frac{\partial \boldsymbol{S}}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial w_{d,i}^{(2)}} = (S_i - y_i) \times x_d^{(2)} \tag{14}$$

Softmax wrt. activation $\boldsymbol{a^{(2)}}$

$$\frac{\partial S_i}{\partial a_i^{(2)}} = \frac{\partial \frac{e^{a_i}}{\Sigma^C}}{\partial a_i} = \frac{e^{a_i}\Sigma^C - e^{a_i}e^{a_i}}{(\Sigma^C)^2} = \frac{e^{a_i}}{\Sigma^C}\frac{\Sigma^C - e^{a_i}}{\Sigma^C} = \frac{e^{a_i}}{\Sigma^C}\left(1 - \frac{e^{z_i}}{\Sigma^C}\right) = S_i\left(1 - S_i\right) \tag{15}$$

$$\frac{\partial S_i}{\partial a_j^{(2)}} = \frac{\partial \frac{e^{a_i}}{\Sigma^C}}{\partial a_j} = \frac{0 - e^{a_i}e^{a_j}}{(\Sigma^C)^2} = -\frac{e^{a_i}}{\Sigma^C}\frac{e^{a_j}}{\Sigma^C} = -S_i S_j \tag{16}$$

Where $\Sigma^C = \Sigma_{j=1}^{C} e^{a_j}$

Cross-entropy wrt. activation $\boldsymbol{a_i^{(2)}}$

$$\begin{aligned}
\frac{\partial L}{\partial a_i^{(2)}} &= -\sum_{j=1}^{C} \frac{\partial y_j \log\left(S_j\right)}{\partial a_i} = -\sum_{j=1}^{C} y_j \frac{\partial \log\left(S_j\right)}{\partial a_i} = -\sum_{j=1}^{C} y_j \frac{1}{S_j}\frac{\partial S_j}{\partial a_i} \\
&= -\frac{y_i}{S_i}\frac{\partial S_i}{\partial a_i} - \sum_{j\neq i}\frac{y_j}{S_j}\frac{\partial S_j}{\partial a_i} = -\frac{y_i}{S_i}S_i\left(1 - S_i\right) - \sum_{j\neq i}^{C}\frac{y_j}{S_j}\left(-S_j S_i\right) \\
&= -y_i + y_i S_i + \sum_{j\neq i} a_j S_i = -y_i + \sum_{j=1}^{C} y_j S_i = -y_i + S_i \sum_{j=1}^{C} y_j \\
&= S_i - y_i
\end{aligned} \tag{17}$$

Activation $\boldsymbol{a_i^{(2)}}$ wrt. weight $\boldsymbol{w_{d,i}^{(2)}}$

$$\frac{\partial a_i^{(2)}}{\partial w_{d,i}^{(2)}} = x_d^{(2)} \tag{18}$$

3.3 Gradient derivation - Dense-512 layer

$$\frac{\partial L}{\partial w_{r,d}^{(1)}} = \frac{\partial L}{\partial \boldsymbol{S}} \frac{\partial \boldsymbol{S}}{\partial \boldsymbol{a}^{(2)}} \frac{\partial \boldsymbol{a}^{(2)}}{\partial x_d^{(2)}} \frac{\partial x_d^{(2)}}{\partial a_d^{(1)}} \frac{\partial a_d^{(1)}}{\partial w_{r,d}^{(1)}} = \underset{1xC}{(\boldsymbol{S} - \boldsymbol{y})} \cdot \underset{Cx1}{\boldsymbol{w_d}^{(2)}} \times \delta(a_d^{(1)}) \times x_r^{(1)} \tag{19}$$

Activations $\boldsymbol{a^{(2)}}$ wrt. Dense-512 output

$$\frac{\partial \boldsymbol{a}^{(2)}}{\partial x_d^{(2)}} = [w_{d,1}, w_{d,2}, ..., w_{d,C}]^T = \boldsymbol{w_d}^{(2)} \tag{20}$$

Dense-512 output (ReLU) wrt. activation $a_d^{(1)}$

$$\frac{\partial x_d^{(2)}}{\partial a_d^{(1)}} = \delta(a_d^{(1)}) \tag{21}$$

Where $\delta(a_d^{(1)})$ is dirac-delta function

$$\delta(a_d^{(1)}) = \begin{cases} 0 & \text{if } a_d^{(1)} < 0 \\ 1 & \text{otherwise} \end{cases} \tag{22}$$

Activation $a_d^{(1)}$ wrt. weight $w_{d,i}^{(2)}$

$$\frac{\partial a_d^{(1)}}{\partial w_{r,d}^{(1)}} = x_r^{(1)} \tag{23}$$

3.4 Regularised expectation loss

There are multiple objective function, other than cross-entropy, one could choose with DNNs for a classification task. However, as Janocha [10] has pointed, not much attention has been devoted in the literature to study how choice of objective functions in classification affects the training performance and model characteristics of a DNN.

In this coursework we consider a loss function based on L2-norm, Equation 24.

$$L(Y,S) = \sum_{n=1}^{N}\sum_{i=1}^{C}(y_{n,i} - S_{n,i})^2 = \sum_{n=1}^{N}\left(-2\sum_{i=1}^{C} y_{n,i}S_{n,i} + \sum_{i=1}^{C}||y_{n,i}||_2 + \sum_{i=1}^{C}||S_{n,i}||_2\right) \tag{24}$$

This loss function measures Euclidean distance between predicted probabilities of each class and the true class. Optimisation of this loss function, can be though of as **regularised minimisation of expected miss-classification probability**. The power of 2 in $(y_{n,i} - S_{n,i})^2$ has intuitive interpretation that larger classification errors are penalised relatively more.

Such loss function could potentially result in a better DNN performance as we focus on maximising expected probability of good classification rather than probability of correct labeling. This could be especially relevant and useful when input data is noisy, which has been empirically proven by Janocha [10].

Nevertheless, for MNIST-Digit classification task, Cross-Entropy loss function appears to result in better models, when compared to alternative loss functions, Table 2.

	X-Entropy	reg. Expectation	$Hinge^2$	Hinge	Cat. Hinge	KL-div
test accuracy	0.968	0.960	0.956	0.951	0.951	0.965

Table 2: Test accuracy for a 2-Layer Fully-connected network trained with different objective functions on MNIST-Digit dataset.

Derivatives There is no need to adjust neural network architecture when changing cross entropy loss to regularised expectation loss. However, the gradients have to be re-derived.

$$\frac{\partial L}{\partial a_i^{(2)}} = \sum_{j=1}^{C} \frac{\partial (y_j - S_j)^2}{\partial a_i} = \sum_{j=1}^{C} \frac{\partial (y_j^2 - 2y_j S_j + S_j^2)}{\partial a_i} = \sum_{j=1}^{C} \frac{\partial (S_j^2 - 2y_j S_j)}{\partial S_j} \frac{\partial S_j}{\partial a_i} = \\ = 2 \sum_{j=1}^{C} (S_j - y_j) S_j (\mathrm{I}(i=j) - S_i) \tag{25}$$

$$\frac{\partial L}{\partial w_{d,i}^{(2)}} = \frac{\partial L}{\partial \boldsymbol{S}} \frac{\partial \boldsymbol{S}}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial w_{d,i}^{(2)}} = 2 \sum_{j=1}^{C} [(S_j - y_j) S_j (\mathrm{I}(i=j) - S_i)] \times x_d^{(2)} \tag{26}$$

$$\frac{\partial L}{\partial w_{r,d}^{(1)}} = \frac{\partial L}{\partial \boldsymbol{S}} \frac{\partial \boldsymbol{S}}{\partial \boldsymbol{a}^{(2)}} \frac{\partial \boldsymbol{a}^{(2)}}{\partial x_d^{(2)}} \frac{\partial x_d^{(2)}}{\partial a_d^{(1)}} \frac{\partial a_d^{(1)}}{\partial w_{r,d}^{(1)}} = \left[\frac{\partial L}{\partial a_1^{(2)}}, \frac{\partial L}{\partial a_2^{(2)}}, ..., \frac{\partial L}{\partial a_C^{(2)}} \right] \cdot \underset{Cx1}{\boldsymbol{w_d}^{(2)}} \times \delta(a_d^{(1)}) \times x_r^{(1)} \tag{27}$$

4 Q 4.4

4.1 Convolutional Layer vs. Fully connected layer

Disadvantage

- Lower representative power due to smaller receptive field

Advantages

- Forms spatial features and is suitable for hierarchical learning.
- Network with Conv-layers needs less parameters for the same representational power, than Network with only Fully-connected layers.

4.2 Disadvantage - Lower representative power due to smaller receptive field

Each neuron in a fully connected layer is connected to every neuron in the previous layer. Thus the layer can learn features of the entire input space. On the contrary, in convolutional layers number of connections to a single neuron is specified by a kernel size and for layers with small kernels, the connections are very local. Therefore, a Fully connected layer has greater representative power than a convolutional layer with a small kernel size.

In our Experiment, we consider a single Hidden convolutional layer and one Dense-10 output layer. We want to study how the test accuracy of this network, varies with the size of the receptive field. We enabled padding such that the size of the output of 1-Hidden-layer is the same for each network. Figure 7 shows that test accuracy, thus representative power of a layer, increases as we increase the kernel size. As we increase stride, the connectivity to previous layer becomes sparser and a larger receptive field is even more important.

Of course, in a convolutional layer the kernel size can be arbitrary large, and in fact when kernel size is equal to the input image, and we use no padding, then the convolutional layer is equivalent to a fully connected layer, where number of filters in convolutional layer corresponds to size of fully-connected layer.

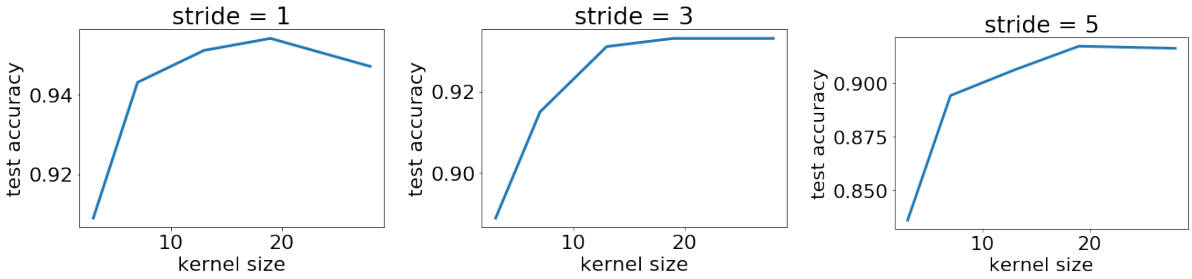


Figure 7: Increase in test accuracy as a result of increased kernel size in a network with a 1-hidden convolutional layer.

4.3 Advantage - Hierarchical learning and spatial features

It is important to note, that previous argument only applies to shallow networks. In Deep Neural Networks we are more concerned about representative power of the entire network rather than of a single layer. Moreover, in a DNN context, a local connectivity of a neuron to the previous layer can be viewed as desirable when working with structured data such as images. There, correlation between pixels and features is strongest locally.

In DNN, the first convolutional layer learns local features of the input, which are then combined by later convolutional layers. As a result, every next layer learns more abstract features, and the effective receptive field with respect to network input increases with the network depth.

Therefore, convolutional layers force extraction of local features in the image at different levels of abstraction (good for structured image data), while the deep architecture makes the receptive field of the network cover the entire image input (necessary for classification). This is unique to convolutional layers and couldn't be achieved with fully connected layers.

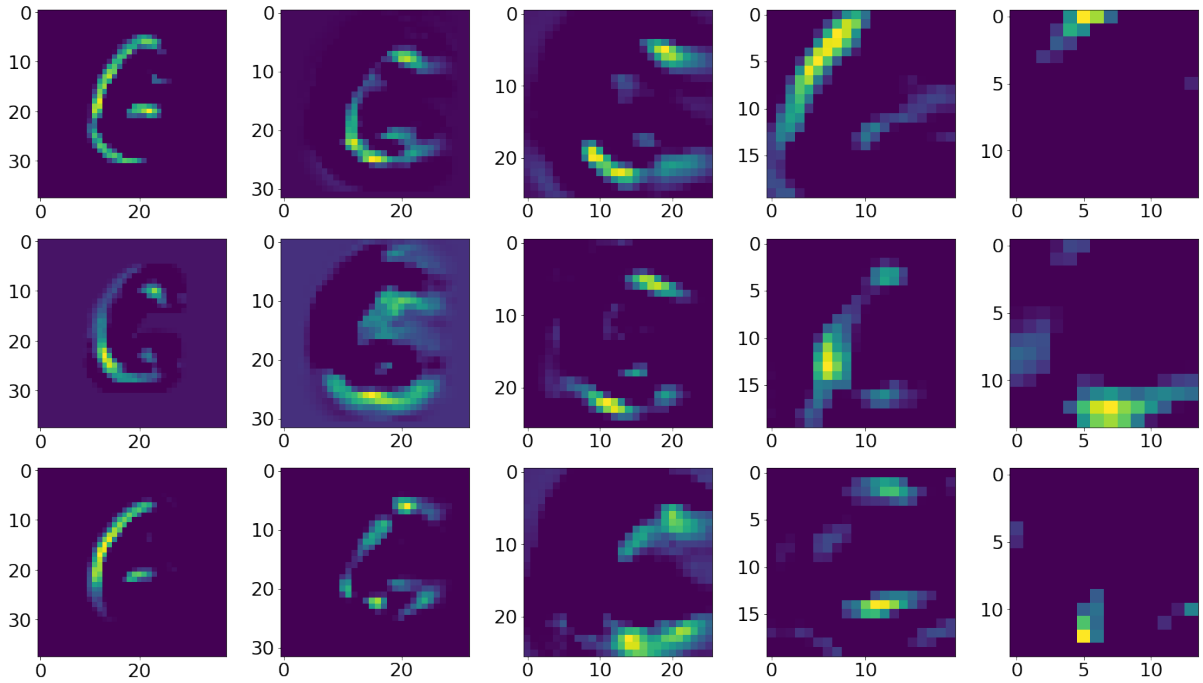
Figure 8 visualises activation values of each of convolutional layers in a simple 5-Hidden-layer network, with kernel size 7 and stride 1. We can observe on Figure 8 that output of early layers is easily interpretable, as the resolution is still high and neurons activate when the filters convolve with digit-like input. For later layers the output is not as easily interpretable as the level of abstraction is large. However, the same principle applies, each next layer reduces resolution and forms some spatial abstraction of features from the previous layer.

4.4 Advantage - Less model parameters

Test accuracy of a network is not the only objective when building NN based classifiers. In addition, we want to minimise the number of parameters in the network, to reduce DRAM memory requirements, as explained in Q4.3.

Convolutional layers have lower number of parameters than fully connected layers, due to local connectivity of neurons with previous layer and due to parameter sharing for neurons within the same channel. Additionally, hierarchical feature learning in DNN with convolutional layers, described in previous section, allows convolutional layer networks to achieve high accuracy despite low number of parameters.

Figure 9 shows how much more parameters fully connected layer needs, to achieve similar test accuracy to that of a 3-Hidden Layer convolutional network. CNN has 3 layers with: kernel size K=5, stride S=2 for two first layers and S=1 for the last, and 6,6,30 channels in each layer, respectively. Thus it uses only 5902 parameters, yet, has better test accuracy then fully-connected network with order of magnitude more parameters.



(a) CONV 1 (b) CONV 2 (c) CONV 3 (d) CONV 4 (e) CONV 5

Figure 8: Visualisation output of 5 convolutional layers for channels 1 (1st row), 2 (2nd row), and 3 (3rd row). Each layer has kernel size of 7 and stride 1. Note dimensions of input images are 38x38 (we enlarged MNIST data set by adding zero rows and columns to make the evolution of the layer output visualisation clearer)

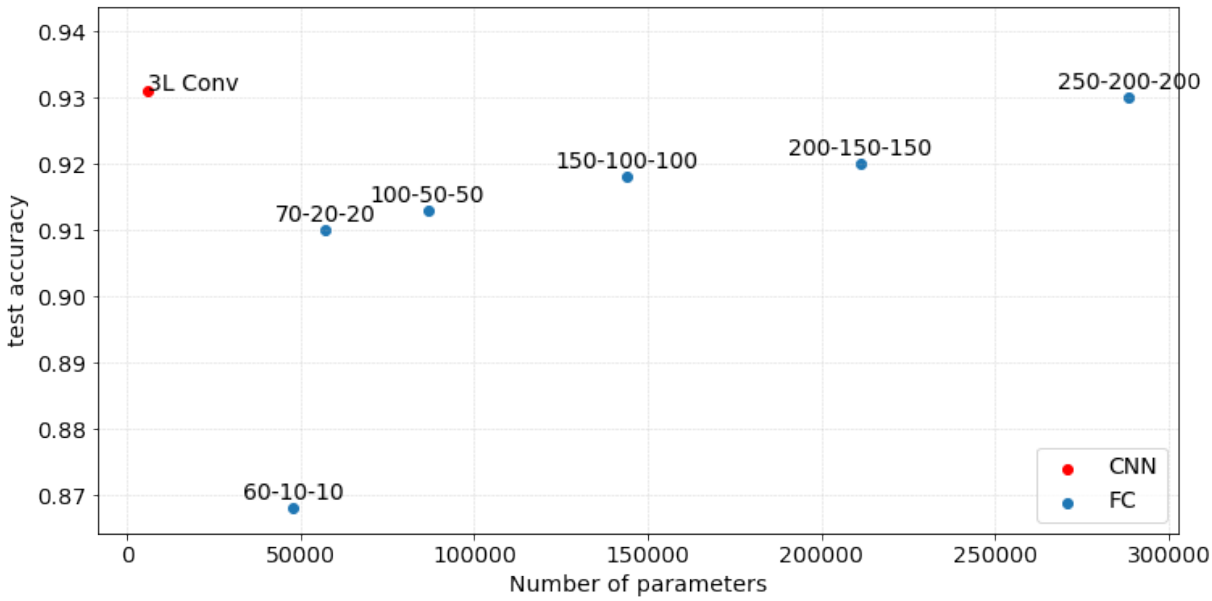


Figure 9: Comparison between model parameters and accuracy of CNN (red) and Fully-connected networks (blue). Annotations: A-B-C indicate number of fully connected neurons at each of the hidden layers. Dropout of 0.2 was used for fully connected networks.

References

[1] Bradski, G. R., and Kaehler, A. (2008). Learning OpenCV. Beijing, O'Reilly.

[2] Hartley, R. and Zisserman, A. (2017). Multiple view geometry in computer vision. New York: Cambridge University Press.

[3] Emanuele Trucco and Alessandro Verri (2006). Introductory techniques for 3-D computer vision. Pearson Education South Asia: Prentice Hall.

[4] Forsyth, D. and Ponce, J. (2015). Computer vision: a modern approach. Boston: Pearson.

[5] Ziegler, R., Matusik, W., Pfister, H., and McMillan, L. 2003. 3d reconstruction using labeled image regions. In Eurographics Symposium on Geometry Processing, 248–259

[6] Paris, Sylvain. (2004). Extraction of Three-dimensional Information from Images – Application to Computer Graphics.

[7] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision, 47(1-3):7-42, April-June 2002.

[8] K. Kutulakos and S. M. Seitz. A Theory of Shape by Space Carving. International Journal of Computer Vision (IJCV),, 38(3):199–218, 2000.

[9] Kiriakos N. Kutulakos. Approximate N-view stereo. In Proceedings of the European Conference on Computer Vision, pages 67–83, 2000

[10] Janocha, K. and Czarnecki, W.M. (2017). On Loss Functions for Deep Neural Networks in Classification. Schedae Informaticae, [online] 1/2016. Available at: https://arxiv.org/pdf/1702.05659.pdf.