Chapter 1

Background

This thesis combines work in several fields. We provide a background for the most important and relevant fields in this chapter. We first introduce the basics of deep learning, before defining the properties of Wavelet Transforms, and finally we introduce the Scattering Transform, the original inspiration for this thesis.

1.1 Supervised Machine Learning

Consider a sample space over inputs and labels $\mathcal{X} \times \mathcal{Y}$ and a data generating distribution p_{data}. Given a dataset of input-label pairs $\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$ we would like to make predictions about $p_{data}(y|x)$ that generalize well to unseen data. A common way to do this is to build a parametric model to directly estimate this conditional probability. For example, regression asserts the data are distributed according to a function of the inputs plus a noise term ϵ:

$$y = f(x, \theta) + \epsilon \tag{1.1.1}$$

This noise is often modelled as a zero mean Gaussian random variable, $\epsilon \sim \mathcal{N}(0, \sigma^2)$, which means we can write:

$$p_{model}(y|x, \theta) = \mathcal{N}(y;\ f(x, \theta), \sigma^2) \tag{1.1.2}$$

with (θ, σ^2) are the parameters of the model.

We can find point estimates of the parameters by maximizing the likelihood of $p_{model}(y|x, \theta)$ (or equivalently, minimizing the KL-divergence between p_{model} and p_{data} $KL(p_{model}||p_{data})$).

As the data are all i.i.d., we can multiply individual likelihoods, and solve for θ:

$$\begin{aligned}\theta_{MLE} &= \arg\max_{\theta} p_{model}(y|x,\theta) && (1.1.3)\\ &= \arg\max_{\theta} \prod_{n=1}^{N} p_{model}(y^{(n)}|x^{(n)},\theta) && (1.1.4)\\ &= \arg\max_{\theta} \sum_{n=1}^{N} \log p_{model}(y^{(n)}|x^{(n)},\theta) && (1.1.5)\end{aligned}$$

Using the regression model from above, this becomes:

$$\begin{aligned}\theta_{MLE} &= \arg\max_{\theta} \sum_{n=1}^{N} \log p_{model}(y^{(n)}|x^{(n)},\theta) && (1.1.6)\\ &= \arg\max_{\theta} \left(-N\log\sigma - \frac{N}{2}\log(2\pi) - \sum_{n=1}^{N} \frac{\left(y^{(n)} - f(x^{(n)},\theta)\right)^2}{2\sigma^2} \right) && (1.1.7)\\ &= \arg\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \frac{\left(y^{(n)} - f(x^{(n)},\theta)\right)^2}{2} && (1.1.8)\end{aligned}$$

which gives us the well known result that we would like to find parameters that minimize the mean squared error (MSE) between observations y and predictions $\hat{y} = f(x,\theta)$.

For binary classification, ($y \in \{0,1\}$) instead of the model in (1.1.2) we have:

$$p_{model}(y|x,\theta) = \text{Ber}(y;\ \sigma(f(x,\theta))) \tag{1.1.9}$$

where σ is the sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{1.1.10}$$

This expands naturally to multi-class classification ($y \in \{0,1\}^C$) by swapping the Bernoulli distribution for the categorical and the sigmoid for a softmax function, defined by:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{C} e^{z_k}} \tag{1.1.11}$$

If we let $\hat{y}_i = \sigma_i(f(x,\theta))$, this makes (1.1.9):

$$\begin{aligned}p_{model}(y|x,\theta) &= \text{Cat}(y;\ \sigma(f(x,\theta))) && (1.1.12)\\ &= \prod_{c=1}^{C} \prod_{n=1}^{N} \left(\hat{y}_c^{(n)}\right)^{\mathbb{I}(y_c^{(n)}=1)} && (1.1.13)\end{aligned}$$

where $\mathbb{I}(x)$ is the indicator function. As $y_c^{(n)}$ is either 0 or 1, we remove the indicator function. Maximizing this likelihood to find the ML estimate for θ:

$$\theta_{MLE} = \arg\min_{\theta} \prod_{c=1}^{C} \prod_{n=1}^{N} \left(\hat{y}_c^{(n)}\right)^{y_c^{(n)}} \tag{1.1.14}$$

$$= \arg\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \sum_{c=1}^{C} y_c^{(n)} \log \hat{y}_c^{(n)} \tag{1.1.15}$$

which we recognize as the cross-entropy between y and $\hat{y}$.

1.1.1 Priors on Parameters and Regularization

Maximum likelihood estimates for parameters, while straightforward, can often lead to overfitting. A common practice is to regularize learnt parameters θ by putting a prior over them. If we do not have any prior information about what we expect the parameters to be, it is still useful to put an uninformative prior on the weights. For example, if our weights are in the reals, a commonly used prior is a Gaussian.

Let us extend the regression example from above by saying we would like the prior on the weights θ to be a Gaussian, i.e. $p(\theta) = \mathcal{N}(0, \tau^2)$. The corresponding maximum a posteriori (MAP) estimate is then obtained by finding:

$$\theta_{MAP} = \arg\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \frac{\left(y^{(n)} - f(x^{(n)}, \theta)\right)^2}{2} + \lambda ||\theta||_2^2 \tag{1.1.16}$$

where $\lambda = \sigma^2/\tau^2$, which is equivalent to minimizing the MSE with an ℓ_2 penalty on the parameters. λ is often called **weight decay** in the neural network literature, which we will also use in this thesis.

1.1.2 Loss Functions and Minimizing the Objective

It may be useful to rewrite (1.1.16) as an objective function on the parameters $J(\theta)$:

$$J(\theta) = \frac{1}{N} \sum_{n=1}^{N} \frac{\left(y^{(n)} - f(x^{(n)}, \theta)\right)^2}{2} + \lambda ||\theta||_2^2 \tag{1.1.17}$$

$$= L_{data}(y, f(x, \theta)) + L_{reg}(\theta) \tag{1.1.18}$$

where L_{data} is the data loss defined, such as MSE or cross-entropy and L_{reg} is the regularization, such as ℓ_2 or ℓ_1 penalized loss.

Now $\theta_{MAP} = \arg\min J(\theta)$. Finding the minimum of the objective function is task-dependent and is often not straightforward. One commonly used technique is called *gradient descent* (GD). This is straightforward to do as it only involves calculating the gradient at

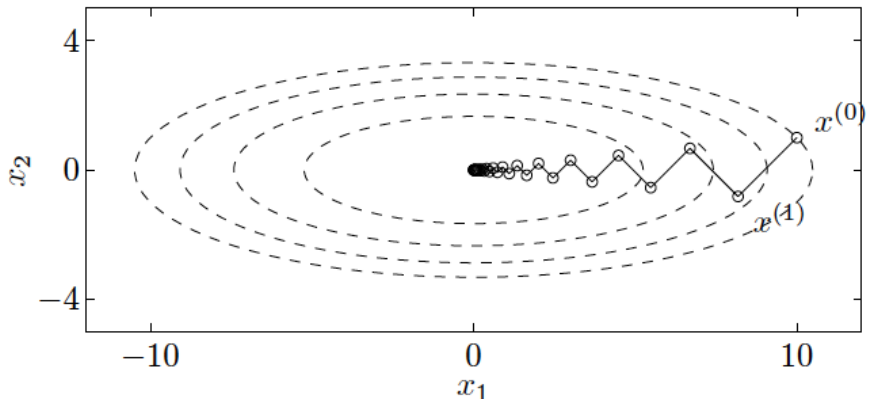

Figure 1.1: **Trajectory of gradient descent in an ellipsoidal parabola.** Some contour lines of the function $f(x) = 1/2\,(x_1^2 + 10x_2^2)$ and the trajectory of GD optimization using exact line search. This space has condition number 10, and shows the slow convergence of GD in spaces with largely different eigenvalues. Image taken from [1] Figure 9.2.

a given point and taking a small step in the direction of steepest descent. The difference equation defining this can be written as:

$$\theta_{t+1} = \theta_t - \eta \frac{\partial J}{\partial \theta} \tag{1.1.19}$$

Unsurprisingly, such a simple technique has limitations. In particular, it has a slow convergence rate when the condition number (ratio of largest to smallest eigenvalues) of the Hessian around the optimal point is large [1]. An example of this is shown in Figure 1.1. In this figure, the step size is chosen with exact line search, i.e.

$$\eta = \arg\min_s f(x + s\frac{\partial f}{\partial x}) \tag{1.1.20}$$

To truly overcome this problem, we must know the curvature of the objective function $\frac{\partial^2 J}{\partial \theta^2}$. An example optimization technique that uses the second order information is Newton's method. Such techniques sadly do not scale with size, as computing the Hessian is proportional to the number of parameters squared, and most neural networks have hundreds of thousands, if not millions of parameters. In this thesis, we only consider *first-order optimization* algorithms.

1.1.3 Stochastic Gradient Descent

Aside from the problems associated the curvature of the function $J(\theta)$, another common issue faced with the gradient descent of (1.1.19) is the cost of computing $\frac{\partial J}{\partial \theta}$. In particular,

the first term:

$$L_{data}(y, f(x,\theta)) = \mathbb{E}_{x,y\sim p_{data}}\left[L_{data}(y, f(x,\theta))\right] \tag{1.1.21}$$

$$= \frac{1}{N}\sum_{n=1}^{N} L_{data}\left(y^{(n)}, f(x^{(n)},\theta)\right) \tag{1.1.22}$$

involves evaluating the entire dataset at the current values of θ. As the training set size grows into the thousands or millions of examples, this approach becomes prohibitively slow.

(1.1.21) writes the data loss as an expectation, hinting at the fact that we can remedy this problem by using fewer samples $N_b < N$ to evaluate L_{data}. This variation is called Stochastic Gradient Descent (SGD).

Choosing the batch size is a hyperparameter choice that we must think carefully about. Setting the value very low, e.g. $N_b = 1$ can be advantageous as the noisy estimates for the gradient have a regularizing effect on the network [2]. Increasing the batch size to larger values allows you to easily parallelize computation as well as increasing your accuracy for the gradient, allowing you to take larger step sizes [3]. A good initial starting point is to set the batch size to about 100 samples and increase/decrease from there [4].

1.1.4 Gradient Descent and Learning Rate

The step size parameter, η in (1.1.19) is commonly referred to as the learning rate. Choosing the right value for the learning rate is key. Unfortunately, the line search algorithm in (1.1.20) would be too expensive to compute for neural networks (as would involve evaluating the function several times at different values), each of which takes about as long as calculating the gradients themselves. Additionally, as the gradients are typically estimated over a mini-batch and are hence noisy there may be little added benefit in optimizing the step sizes in the estimated direction.

Figure 1.2 illustrates the effect the learning rate can have over a contrived convex example. Optimizing over more complex loss surfaces only exacerbates the problem. Sadly, choosing the initial learning rate is 'more of an art than a science' [4], but [5], [6] have some tips on what to set this at. We have found in our work that searching for a large learning rate that causes the network to diverge and reducing it hence can be a good search strategy. This agrees with Section 1.5 of [7] which states that for regions of the loss space which are roughly quadratic, $\eta_{max} = 2\eta_{opt}$ and any learning rate above $2\eta_{opt}$ causes divergence.

On top of the initial learning rate, the convergence of SGD methods require:

$$\sum_{t=1}^{\infty}\eta_t \rightarrow \infty \tag{1.1.23}$$

$$\sum_{t=1}^{\infty}\eta_t^2 = M \tag{1.1.24}$$

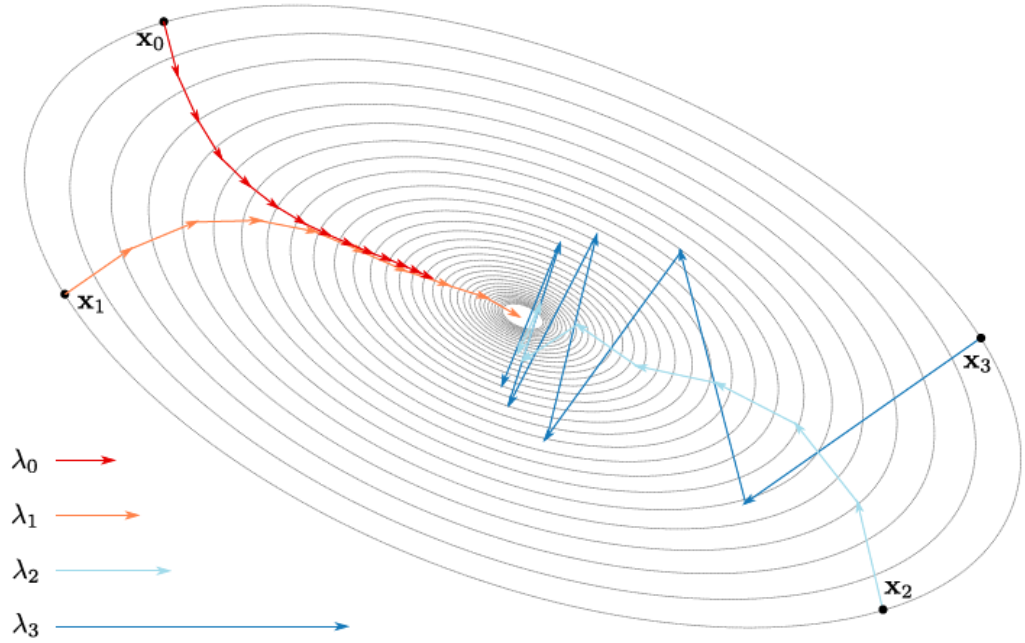

Figure 1.2: **Trajectories of SGD with different initial learning rates.** This figure illustrates the effect the step size has over the optimization process by showing the trajectory for $\eta = \lambda_i$ from equivalent starting points on a symmetric loss surface. Increasing the step size beyond λ_3 can cause the optimization procedure to diverge. Image taken from [8] Figure 2.7.

where M is finite. Choosing how to do this also contains a good amount of artistry, and there is no one scheme that works best. A commonly used greedy method is to keep the learning rate constant until the training loss stabilizes and then to enter the next phase of training by setting $\eta_{k+1} = \gamma\eta_k$ where γ is a decay factor. Choosing γ and the thresholds for triggering a step however must be chosen by monitoring the training loss curve and trial and error.

1.1.5 Momentum and Adam

One simple and very popular modification to SGD is to add *momentum*. Momentum accumulates past gradients with an exponentially moving average and continues to move in their direction. The name comes from the analogy of finding a minimum of a function to rolling a ball over a loss surface – any new force (newly computed gradients) must overcome the past motion of the ball. To do this, we create a *velocity* variable v_t and modify (1.1.19)

to be:

$$v_{t+1} = \alpha v_t - \eta_k \frac{\partial J}{\partial \theta} \tag{1.1.25}$$
$$\theta_{t+1} = \theta_t + v_{t+1} \tag{1.1.26}$$
$$\tag{1.1.27}$$

where $0 \leq \alpha < 1$ is the momentum term indicating how quickly to 'forget' past gradients.

Another popular modification to SGD is the adaptive learning rate technique Adam [9]. There are several other adaptive schemes such as AdaGrad [10] and AdaDelta [11], but they are all quite similar, and Adam is often considered the most robust of the three [4]. The goal of all of these adaptive schemes is to take larger update steps in directions of low variance, helping to minimize the effect of large condition numbers we saw in Figure 1.1. Adam does this by keeping track of the first m_t and second v_t moments of the gradients:

$$g_{t+1} = \frac{\partial J}{\partial \theta} \tag{1.1.28}$$
$$m_{t+1} = \beta_1 m_t + (1-\beta_1) g_{t+1} \tag{1.1.29}$$
$$v_{t+1} = \beta_2 m_t + (1-\beta_2) g_{t+1} \tag{1.1.30}$$

where $0 \leq \beta_1, \beta_2 < 1$. Note the similarity between updating the mean estimate in (1.1.29) and the velocity term in (1.1.25)[1]. The parameters are then updated with:

$$\theta_{t+1} = \theta_t - \eta \frac{m_{t+1}}{\sqrt{v_{t+1}} + \epsilon} \tag{1.1.31}$$

where ϵ is a small value to avoid dividing by zero.

1.2 Neural Networks

1.2.1 The Neuron and Single Layer Neural Networks

The neuron, shown in Figure 1.3 is the core building block of Neural Networks. It takes the dot product between an input vector $\mathbf{x} \in \mathbb{R}^D$ and a weight vector $\mathbf{w}$, before applying a chosen nonlinearity, g. I.e.

$$y = g(\langle \mathbf{x}, \mathbf{w} \rangle) = g\left(\sum_{i=0}^{D} x_i w_i\right) \tag{1.2.1}$$

[1] The m_{t+1} and v_{t+1} terms are then bias-corrected as they are biased towards zero at the beginning of training. We do not include this for conciseness.

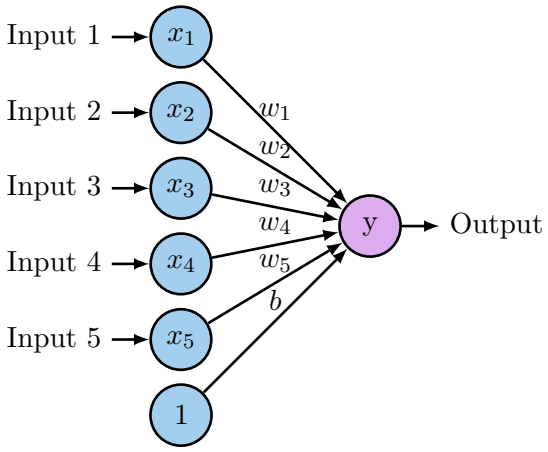

Figure 1.3: **A single neuron.** The neuron is composed of inputs x_i, weights w_i (and a bias term), as well as an activation function. Typical activation functions include the sigmoid function, tanh function and the ReLU

where we have used the shorthand $b = w_0$ and $x_0 = 1$. Also note that we will use the neural network common practice of calling the *weights* w, compared to the parameters θ we have been discussing thus far.

Typical nonlinear functions g are the sigmoid function (already presented in (1.1.10)), but also common are the tanh and ReLU functions:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.2.2}$$

$$\text{ReLU}(x) = \max(x, 0) \tag{1.2.3}$$

See Figure 1.4 for plots of these. The original Rosenblatt perceptron [12] used the Heaviside function $\text{H}(x) = \mathbb{I}(x > 0)$.

Note that if $\langle \mathbf{w}, \mathbf{w} \rangle = 1$ then $\langle \mathbf{x}, \mathbf{w} \rangle$ is the distance from the point $\mathbf{x}$ to the hyperplane with normal $\mathbf{w}$. With general $\mathbf{w}$ this can be thought of as a scaled distance. Thus, the weight vector $\mathbf{w}$ defines a hyperplane in $\mathbb{R}^D$ which splits the space into two. The choice of nonlinearity then affects how points on each side of the plane are treated. For a sigmoid, points far below the plane get mapped to 0 and points far above the plane get mapped to 1 (with points near the plane having a value of 0.5). For tanh nonlinearities, these points get mapped to -1 and 1. For ReLU nonlinearities, every point below the plane ($\langle \mathbf{x}, \mathbf{w} \rangle < 0$) gets mapped to zero and every point above the plane keeps its inner product value.

Nearly all modern neural networks use the ReLU nonlinearity and it has been credited with being a key reason for the recent surge in deep learning success [13], [14]. In particular:

1. It is less sensitive to initial conditions as the gradients that backpropagate through it will be large even if x is large. A common observation of sigmoid and tanh non-linearities was that their learning would be slow for quite some time until the neurons came out

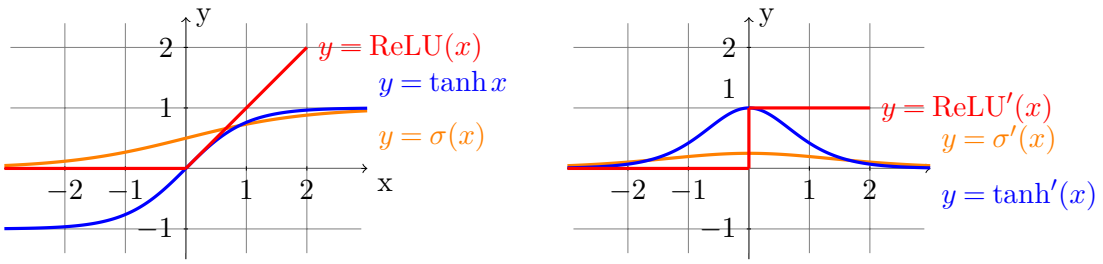

Figure 1.4: **Common Neural Network nonlinearities and their gradients.** The sigmoid, tanh and ReLU nonlinearities are commonly used activation functions for neurons. Note the different properties. In particular, the tanh and sigmoid have the nice property of being smooth but can have saturation when the input is either largely positive or largely negative, causing little gradient to flow back through it. The ReLU does not suffer from this problem, and has the additional nice property of setting values to exactly 0, making a sparser output activation.

of saturation, and then their accuracy would increase rapidly before levelling out again at a minimum [15]. The ReLU, on the other hand, has constant gradient.

2. It promotes sparsity in outputs, by setting them to a hard 0. Studies on brain energy expenditure suggest that neurons encode information in a sparse manner. [16] estimates the percentage of neurons active at the same time to be between 1 and 4%. Sigmoid and tanh functions will typically have *all* neurons firing, while the ReLU can allow neurons to fully turn off.

1.2.2 Multilayer Perceptrons

As mentioned in the previous section, a single neuron can be thought of as a separating hyperplane with an activation that maps the two halves of the space to different values. Such a linear separator is limited, and famously cannot solve the XOR problem [17]. Fortunately, adding a single hidden layer like the one shown in Figure 1.5 can change this, and it is provable that with an infinitely wide hidden layer, a neural network can approximate any function [18], [19].

The forward pass of such a network with one hidden layer of H units is:

$$h_i = g\left(\sum_{j=0}^{D} x_j w_{ij}^{(1)}\right) \tag{1.2.4}$$

$$y = \sum_{k=0}^{H} h_k w_k^{(2)} \tag{1.2.5}$$

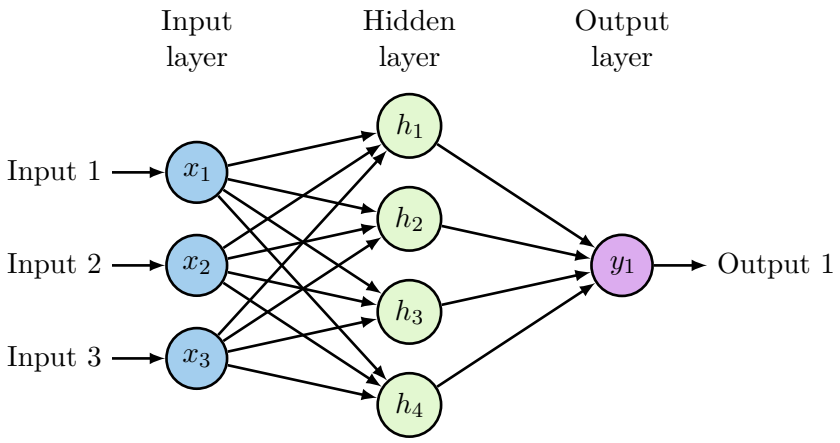

Figure 1.5: **Multi-layer perceptron.** Expanding the single neuron from Figure 1.3 to a network of neurons. The internal representation units are often referred to as the *hidden layer* as they are an intermediary between the input and output.

where $w^{(l)}$ denotes the weights for the l-th layer, of which Figure 1.5 has 2. Note that these individual layers are often called *fully connected* as each node in the previous layer affects every node in the next.

If we were to expand this network to have L such fully connected layers, we could rewrite the action of each layer in a recursive fashion:

$$Y^{(l+1)} = W^{(l+1)} X^{(l)} \tag{1.2.6}$$

$$X^{(l+1)} = g\left(Y^{(l+1)}\right) \tag{1.2.7}$$

where W is now a weight matrix, acting on the vector of previous layer's outputs $X^{(l)}$. As we are now considering every layer an input to the next stage, we have removed the h notation, and added the superscript (l) to define the depth. $X^{(0)}$ is the network input and $Y^{(L)}$ is the network output. Let us say that the output has C nodes, and a hidden layer $X^{(l)}$ has C_l nodes.

1.2.3 Backpropagation

It is important to truly understand backpropagation when designing neural networks, so we describe the core concepts now for a neural network with L layers.

The delta rule, initially designed for networks with no hidden layers [20], was expanded to what we now consider *backpropagation* in [21]. While backpropagation is conceptually just the application of the chain rule, Rumelhart, Hinton, and Williams successfully updated the delta rule to networks with hidden layers, laying a key foundational step for deeper networks.

With a deep network, calculating $\frac{\partial J}{\partial w}$ may not seem particularly obvious if w is a weight in one of the earlier layers. We need to define a rule for updating the weights in all L layers

of the network, $W^{(1)}, W^{(2)}, \dots W^{(L)}$ however, only the final set $W^{(L)}$ are connected to the objective function J.

1.2.3.1 Regression Loss

Let us start with writing down the derivative of J with respect to the network output $Y^{(L)}$ using the regression objective function (1.1.8). As we now have two superscripts, one for the sample number and one for the layer number, we combine them into a tuple of superscripts.

$$\begin{aligned} \frac{\partial J}{\partial Y^{(L)}} &= \frac{\partial}{\partial Y^{(L)}} \left(\frac{1}{N} \sum_{n=1}^{N_b} \frac{1}{2} \left(y^{(n)} - Y^{(L,n)} \right)^2 \right) && (1.2.8) \\ &= \frac{1}{N} \sum_{n=1}^{N_b} \left(Y^{(L,n)} - y^{(n)} \right) && (1.2.9) \\ &= e \in \mathbb{R} && (1.2.10) \end{aligned}$$

where we have used the fact that for the regression case, $y^{(n)}, Y^{(L,n)} \in \mathbb{R}$.

1.2.3.2 Classification Loss

For the classification case (1.1.15), let us keep the output of the network $Y^{(L,n)} \in \mathbb{R}^C$ and define an intermediate value $\hat{y}$ the softmax applied to this vector $\hat{y}_c^{(n)} = \sigma_c \left(Y^{(L,n)} \right)$. Note that the softmax is a vector valued function going from $\mathbb{R}^C \to \mathbb{R}^C$ so has a jacobian matrix $S_{ij} = \frac{\partial \hat{y}_i}{\partial Y_j^{(L)}}$ with values:

$$S_{ij} = \begin{cases} \sigma_i(1 - \sigma_j) & \text{if } i = j \\ -\sigma_i \sigma_j & \text{if } i \neq j \end{cases} \tag{1.2.11}$$

Now, let us return to (1.1.15) and find the derivative of the objective function to this intermediate value $\hat{y}$:

$$\begin{aligned} \frac{\partial J}{\partial \hat{y}} &= \frac{\partial}{\partial \hat{y}} \left(\frac{1}{N} \sum_{n=1}^{N_b} \sum_{c=1}^{C} y_c^{(n)} \log \hat{y}_c^{(n)} \right) && (1.2.12) \\ &= \frac{1}{N} \sum_{n=1}^{N_b} \sum_{c=1}^{C} \frac{y_c^{(n)}}{\hat{y}_c^{(n)}} && (1.2.13) \\ &= d \in \mathbb{R}^C && (1.2.14) \end{aligned}$$

Note that unlike (1.2.10), this derivative is vector valued. To find $\frac{\partial J}{\partial Y^{(L)}}$ we use the chain rule. It is easier to find the partial derivative with respect to one node in the output first, and then expand from here. I.e.:

$$\frac{\partial J}{\partial Y_j^{(L)}} = \sum_{i=1}^{C} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial Y_j^{(L)}} \tag{1.2.15}$$

$$= S_j^T d \tag{1.2.16}$$

where S_j is the jth column of the jacobian matrix S. It becomes clear now that to get the entire vector derivative for all nodes in $Y^{(L)}$, we must multiply the transpose of the jacobian matrix with the error term from (1.2.14):

$$\frac{\partial J}{\partial Y^{(L)}} = S^T d \tag{1.2.17}$$

1.2.3.3 Final Layer Weight Gradient

Let us continue by assuming $\frac{\partial J}{\partial Y^{(L)}}$ is vector valued as was the case with classification. For regression, it is easy to set $C = 1$ in the following to get the necessary results. Additionally for clarity, we will drop the layer superscript in the intermediate calculations.

We call the gradient for the final layer weights the *update* gradient. It can be computed by the chain rule again:

$$\frac{\partial J}{\partial W_{ij}} = \frac{\partial J}{\partial Y_i} \frac{\partial Y_i}{\partial W_{ij}} + 2\lambda W_{ij} \tag{1.2.18}$$

$$= \frac{\partial J}{\partial Y_i} X_j + 2\lambda W_{ij} \tag{1.2.19}$$

where the second term in the above two equations comes from the regularization loss that is added to the objective. The gradient of the entire weight matrix is then:

$$\frac{\partial J}{\partial W^{(L)}} = \frac{\partial J}{\partial \hat{y}} X^T + 2\lambda W \tag{1.2.20}$$

$$= S^T d \left(X^{(L-1)} \right)^T + 2\lambda W^{(L)} \in \mathbb{R}^{C \times C_{L-1}} \tag{1.2.21}$$

1.2.3.4 Final Layer Passthrough Gradient

Additionally, we want to find the *passthrough* gradients of the final layer $\frac{\partial J}{\partial X^{(L-1)}}$. In a similar fashion, we first find the gradient with respect to individual elements in $X^{(L-1)}$ before

generalizing to the entire vector:

$$\frac{\partial J}{\partial X_i} = \sum_{j=1}^{C} \frac{\partial J}{\partial Y_j}\frac{\partial Y_j}{\partial X_i} \tag{1.2.22}$$

$$= \sum_{j=1}^{C} \frac{\partial J}{\partial Y_j} W_{j,i} \tag{1.2.23}$$

$$= W_i^T \frac{\partial J}{\partial Y} \tag{1.2.24}$$

$$\tag{1.2.25}$$

where W_i is the ith column of W. Thus

$$\frac{\partial J}{\partial X^{(L-1)}} = \left(W^{(L)}\right)^T \frac{\partial J}{\partial Y^{(L)}} \tag{1.2.26}$$

$$= \left(W^{(L)}\right)^T S^T d \tag{1.2.27}$$

This passthrough gradient then can be used to update the next layer's weights by repeating subsubsection 1.2.3.3 and subsubsection 1.2.3.4.

1.2.3.5 General Layer Update

The easiest way to handle this flow of gradients, and the basis for most automatic differentiation packages, is the block definition shown in Figure 1.6. For all neural network components (even if they do not have weights), the operationg must not only be able to calculate the forward pass $y = f(x, w)$ given weights w and inputs x, but also calculate the *update* and *passthrough* gradients $\frac{\partial J}{\partial w}, \frac{\partial J}{\partial x}$ given an input gradient $\frac{\partial J}{\partial y}$. The input gradient will have the same shape as y as will the update and passthrough gradients match the shape of w and x. This way, gradients for the entire network cam be computed in an iterative fashion starting at the loss function and moving backwards.

1.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a special type of of Neural Network where the weights of the fully connected layer are shared across the layer. In this way, a neuron at a given layer is only affected by nodes from the previous layer in a given neighbourhood rather than every node.

First popularized in 1998 by LeCun et. al in [22], the convolutional layer was introduced to build invariance with respect to translations, as well as reduce the parameter size of early neural networks for pattern recognition. The idea of having a locally receptive field had already been shown to be a naturally occurring phenomen by Hubel and Wiesel [23]. They did not become popular immediately, and another spatially based keypoint extractor,

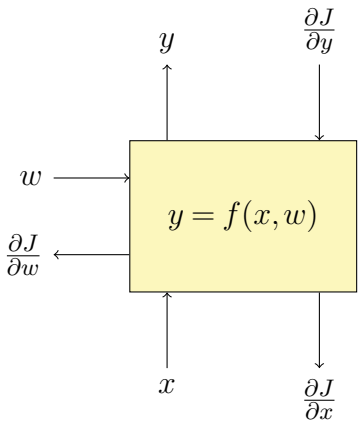

Figure 1.6: **General block form for autograd.** All neural network functions need to be able to calculate the forward pass $y = f(x, w)$ as well as the update and passthrough gradients $\frac{\partial J}{\partial w}, \frac{\partial J}{\partial x}$. Backpropagation is then easily done by allowing data to flow backwards through these blocks from the loss.

SIFT [24], was the mainstay of detection systems until the AlexNet CNN [25] won the 2012 ImageNet challenge [26] by a large margin over the next competitors who used SIFT and Support Vector Machines [27]. This CNN had 5 convolutional layers followed by 3 fully connected layers.

We will briefly describe the convolutional layer, as well as many other layers that have become popular in the past few years.

1.3.1 Convolutional Layers

In the presentation of neural networks so far, we have considered column vectors $X^{(l)}, Y^{(l)} \in \mathbb{R}^{C_l}$. Convolutional layers for image analysis have a different format. In particular, the spatial component of the input is preserved.

Let us first consider the definition of 2-D convolution for single channel images:

$$y[\mathbf{n}] = (x * h)[\mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}] h[\mathbf{n} - \mathbf{k}] \tag{1.3.1}$$

$$= \sum_{k_1, k_2} x[k_1, k_2] h[n_1 - k_1, n_2 - k_2] \tag{1.3.2}$$

where the sum is done over the support of h. For an input $x \in \mathbb{R}^{H \times W}$ and filter $h \in \mathbb{R}^{K_1 \times K_2}$ the output has spatial support $y \in \mathbb{R}^{H+K_1-1 \times W+K_2-1}$.

In the context of convolutional layers, this filter h is a *matched filter* that gives its largest output when the input contains h. If the input has shapes similar to h in many locations, each of these locations in y will also have large outputs.

It is not enough to only have a single matched filter, often we would like to have a bank of them, each sensitive to different shapes. For example, if h_1 was sensitive to horizontal edges, we may also want to detect vertical and diagonal edges. Without specifying what each of the filters do, we can however specify that we would like to detect C different shapes over the spatial extent of an input.

This then means that we have C output channels:

$$\begin{aligned} y_1[\mathbf{n}] &= (x * h_1)[\mathbf{n}] \\ y_2[\mathbf{n}] &= (x * h_1)[\mathbf{n}] \\ &\vdots \\ y_C[\mathbf{n}] &= (x * h_1)[\mathbf{n}] \end{aligned}$$

If we stack red, green and blue input channels on top of each other, we have a 3-dimensional array $x \in \mathbb{R}^{C \times H \times W}$ with $C = 3$.[2] In a CNN layer, each filter h is 3 dimensional with spatial extent exactly equal to C. The *convolution* is done over the remaining two dimensions and the C outputs are summed at each pixel location. This makes (1.3.1):

$$y[\mathbf{n}] = \sum_{c=1}^{C} \sum_{\mathbf{k}} x[\mathbf{k}] h[c, \mathbf{n} - \mathbf{k}] \tag{1.3.3}$$

Again, we would like to have many matched filters to find different shapes in the previous layer, so we repeat Equation 1.3.3 F times and stack the output to give $y \in \mathbb{R}^{F \times H \times W}$:

$$y[f, \mathbf{n}] = \sum_{c=1}^{C} \sum_{\mathbf{k}} x[c, \mathbf{k}] h_f[c, \mathbf{n} - \mathbf{k}] \tag{1.3.4}$$

After a convolutional layer, we can then apply a pointwise nonlinearity g to each output location in y. Revisiting (1.2.6) and (1.2.7), we can rewrite this for a convolutional layer at depth l with C_l input and C_{l+1} output channels:

$$Y^{(l+1)}[f, \mathbf{n}] = \sum_{c=1}^{C_l} X^{(l)}[c, \mathbf{n}] * h_f^{(l)}[c, \mathbf{n}] \qquad \text{for } 1 \leq f \leq C_{l+1} \tag{1.3.5}$$

$$X^{(l+1)}[f, \mathbf{n}] = g\left(Y^{(l)}[f, \mathbf{n}]\right) \tag{1.3.6}$$

This is shown in Figure 1.7.

[2] In deep learning literature, there is not consensus about whether to stack the outputs with the channel first ($\mathbb{R}^{C \times H \times W}$) or last ($\mathbb{R}^{H \times W \times C}$). The latter is more common in Image Processing for colour and spectral images, however the former is the standard for the deep learning framework we use – PyTorch [28], so we use this in this thesis.

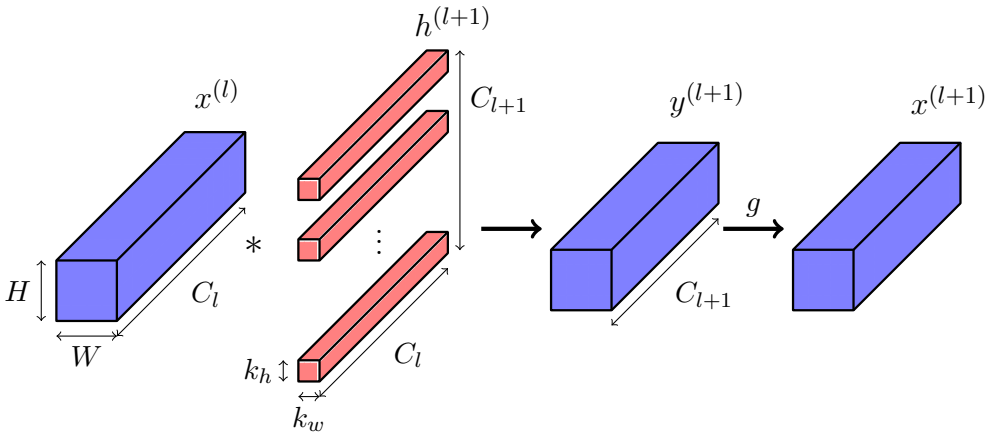

Figure 1.7: **A convolutional layer.** A convolutional layer followed by a nonlinearity g. The previous layer's activations are convolved with a bank of C_{l+1} filters, each of which has spatial size $k_h \times k_w$ and depth C_l. Note that there is no convolution across the channel dimension. Each filter produces one output channel in $y^{(l+1)}$.

1.3.1.1 Padding and Stride

Regular 2-D convolution expands the input from size $H \times W$ to $(H + K_H - 1) \times (W + K_W - 1)$. In convolutional layers in neural networks, it is often desirable and common to have the same output size as input size. This is achieved by taking the central $H \times W$ outputs. We call this *same-size convolution*. Another option commonly used is to only evaluate the kernels where they fully overlap the input signal, causing a reduction in the output size to $(H - K_H + 1) \times (W - K_W + 1)$. This is called *valid convolution* and was used in the original LeNet-5 [22].

Signal extension is by default *zero padding*, and most deep learning frameworks have no ability to choose other padding schemes as part of their convolution functions. Other padding such as *symmetric padding* can be achieved by expanding the input signal before doing a valid convolution.

Stride is a commonly used term in deep learning literature. A stride of 2 means that we evaluate the filter kernel at every other sample point. In signal processing, this is simply called decimation.

1.3.1.2 Gradients

To get the update and passthrough gradients for the convolutional layer we will need to expand (1.3.5). Again we will drop the layer superscripts for clarity:

$$Y[f, n_1, n_2] = \sum_{c=1}^{C} \sum_{k_1} \sum_{k_2} x[c, k_1, k_2] h_f[c, n_1 - k_1, n_2 - k_2] \tag{1.3.7}$$

It is clear from this that a single activation $X[c, n_1, n_2]$ affects many output values. In particular, the derivative for an activation in Y w.r.t. an activation in X is the sum of all the chain rule applied to all these positions:

$$\frac{\partial Y_{f,n_1,n_2}}{\partial X_{c,k_1,k_2}} = h_f[c, n_1 - k_1, n_2 - k_2] \tag{1.3.8}$$

and the derivative from the loss is then:

$$\begin{aligned} \frac{\partial J}{\partial X_{c,k_1,k_2}} &= \sum_f \sum_{n_1} \sum_{n_2} \frac{\partial J}{\partial Y_{f,n_1,n_2}} \frac{\partial Y_{f,n_1,n_2}}{\partial X_{c,k_1,k_2}} && (1.3.9) \\ &= \sum_f \sum_{n_1} \sum_{n_2} \frac{\partial J}{\partial Y_{f,n_1,n_2}} h_f[c, n_1 - k_1, n_2 - k_2] && (1.3.10) \end{aligned}$$

Now we let $\Delta Y[f, n_1, n_2] = \frac{\partial J}{\partial Y_{f,n_1,n_2}}$ be the passthrough gradient signal from the next layer, and $\tilde{h}_\alpha[\beta, \gamma, \delta] = h_\beta[\alpha, -\gamma, -\delta]$ be a set of filters that have been mirror imaged in the spatial domain and had their channel and filter number ordering swapped. Combining these two and subbing into (1.3.10) we get the *passthrough gradient* for the convolutional layer:

$$\begin{aligned} \frac{\partial J}{\partial X_{c,k_1,k_2}} &= \sum_f \sum_{n_1} \sum_{n_2} \Delta Y[f, n_1, n_2] \tilde{h}_c[f, k_1 - n_1, k_2 - n_2] && (1.3.11) \\ &= \sum_f \Delta Y[f, \mathbf{n}] * \tilde{h}_c[f, \mathbf{n}] && (1.3.12) \end{aligned}$$

which is the same as (1.3.5). I.e. we can backpropagate the gradients through a convolutional block by mirror imaging the filters, transposing them in the channel and filter dimensions, and doing a forward convolutional layer with $\tilde{h}$ applied to ΔY. Similarly, we find the *update gradients* to be:

$$\begin{aligned} \frac{\partial J}{\partial h_{f,c,k_1,k_2}} &= \sum_{n_1} \sum_{n_2} \frac{\partial J}{\partial Y_{f,n_1,n_2}} \frac{\partial Y_{f,n_1,n_2}}{\partial h_{f,c,k_1,k_2}} && (1.3.13) \\ &= \sum_{n_1} \sum_{n_2} \Delta Y[f, n_1, n_2] X[x, n_1 - k_1, n_2 - k_2] && (1.3.14) \\ &= (\Delta Y[f, \mathbf{n}] \star X[c, \mathbf{n}])[k_1, k_2] && (1.3.15) \end{aligned}$$

where $\star$ is the cross-correlation operation.

1.3.2 Pooling

Pooling layers are common in CNNs where we want to reduce the spatial size. As we go deeper into a CNN, it is common for the spatial size of the activation to decrease, and the channel dimension to increase. The C_l values at a given spatial location can then be thought of as a feature vector describing the presence of shapes in a given area in the input image.

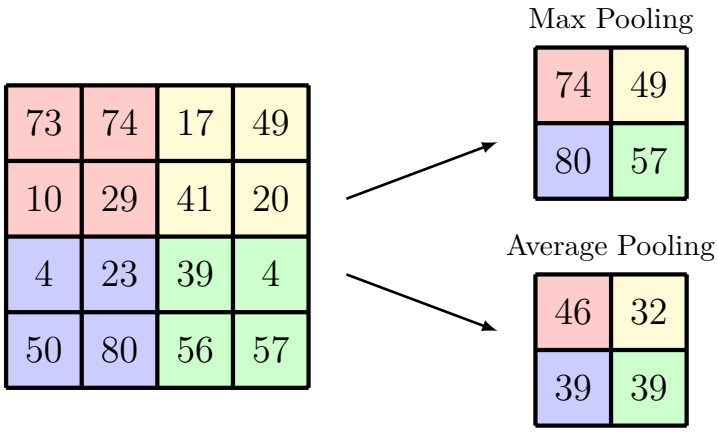

Figure 1.8: **Max vs Average 2×2 pooling.**

Pooling is useful to add some invariance to smaller shifts when downsampling. It is often done over small spatial sizes, such as 2×2 or 3×3. Invariance to larger shifts can be built up with multiple pooling (and convolutional) layers.

Two of the most common pooling techniques are *max pooling* and *average pooling*. Max pooling takes the largest value in its spatial area, whereas average pooling takes the mean. A visual explanation is shown in Figure 1.8. Note that pooling is typically a spatial operation, and only in rare cases is done over the channel dimension.

A review of pooling methods in [29] found them both to perform equally well. While max pooling was the most popular in earlier state of the art networks [25], [30], there has been a recent trend towards using average pooling [31] or even to do away with pooling altogether in favour of strided convolutions (this idea was originally proposed in [32] and used notably in [33]–[35]).

1.3.3 Dropout

Dropout is a particularly harsh regularization scheme that randomly turns off or zeros out neurons in a neural network [36], [37]. Each neuron has probability p of having its value set to 0 during training time, forcing the network to be more general and preventing 'co-adaption' of neurons. The main explanation given in [37] is that dropout averages over several 'thinner' models.

During test time, dropout is typically turned off, but can still be used to get an estimate on the uncertainty of the network by averaging over several runs [38].

1.3.4 Batch Normalization

Batch normalization proposed in [39] is a conceptually simple technique. Despite this, it has become very popular and has been found to be very useful to train *deeper* CNNs.

Batch Normalization rescales CNN activations by channel. Define the mean and standard deviations for a channel across the entire dataset as:

$$\mu_c = \frac{1}{N}\sum_{\mathbf{n}} X^{(n)}[c,\mathbf{n}] \tag{1.3.16}$$

$$\sigma_c^2 = \frac{1}{N}\sum_{\mathbf{n}} \left(X^{(n)}[c,\mathbf{n}]\right)^2 - \mu_c^2 \tag{1.3.17}$$

where $\mu, \sigma \in \mathbb{R}^C$. Batch norm removes the natural mean and variance of the data, scales the data by a learnable gain γ, and shifts it to a learnable position β, with $\gamma, \beta \in \mathbb{R}^C$:

$$Y[c,\mathbf{n}] = \frac{X[c,\mathbf{n}] - \mu_c}{\sigma_c + \epsilon}\gamma_c + \beta_c \tag{1.3.18}$$

where ϵ is a small value to avoid dividing by 0.

Of course, during training, we do not have access to the dataset μ, σ and these values must be estimated from the batch statistics. A typical practice is to keep an exponential moving average estimate of these values to give us $\tilde{\mu}, \tilde{\sigma}$.

The passthrough and update gradients are:

$$\frac{\partial J}{\partial X_{c,n_1,n_2}} = \frac{\partial J}{\partial Y_{c,n_1,n_2}}\frac{\gamma}{\sigma+\epsilon} \tag{1.3.19}$$

$$\frac{\partial J}{\partial \beta_c} = \sum_{\mathbf{n}} \frac{\partial J}{\partial Y_{c,\mathbf{n}}} \tag{1.3.20}$$

$$\frac{\partial J}{\partial \gamma_c} = \sum_{\mathbf{n}} \frac{\partial J}{\partial Y_{c,\mathbf{n}}}\frac{X_{c,\mathbf{n}} - \mu_c}{\gamma_c + \epsilon} \tag{1.3.21}$$

Batch normalization layers are typically placed *between* convolutional layers and non-linearities. I.e. consider the input $X = WU$ for some linear operation on the previous layer's activations U with weights W.

We see that it has the particular benefit of removing the sensitivity of our network initial weight scale, as on the forward pass $BN(aWU) = BN(WU)$. It is also particularly useful for backpropagation, as an increase in weights leads to *smaller* gradients [39], making the network far more resilient to the problems of vanishing and exploding gradients:

$$\begin{aligned} \frac{\partial BN((aW)U)}{\partial U} &= \frac{\partial BN(WU)}{\partial u} \\ \frac{\partial BN((aW)U)}{\partial (aW)} &= \frac{1}{a}\cdot\frac{\partial BN(WU)}{\partial W} \end{aligned} \tag{1.3.22}$$

1.4 Relevant Architectures

In this section we briefly review some relevant CNN architectures that will be helpful to refer back to in this thesis.

1.4.1 Datasets

When doing image analysis tasks, it is important to know comparatively how well different networks perform on the same challenege. To achieve this, the community has developed several datasets that are commonly used to report metrics. For image classification, there are five such datasets, listed here in increasing order of difficulty:

1. **MNIST**: 10 classes, 6000 images per class, 28×28 pixels per image. The images contain the digits 0–9 in greyscale on a blank background. The digits have been size normalized and centred. Dataset description and files can be obtained at [40].

2. **CIFAR-10**: 10 classes, 5000 images per class, 32×32 pixels per image. The images contain classes of everyday objects like cars, dogs, planes etc. The images are colour and have little clutter or background. Dataset description can be found in [41] and files at [42].

3. **CIFAR-100**: 100 classes, 500 images per class, 32×32 pixels per image. Similar to CIFAR-10, but now with fewer images per class and ten times as many classes. Dataset description can be found in [41] and files at [42].

4. **Tiny ImageNet**: 200 classes, 500 images per class, 64×64 pixels per image. A more recently introduced dataset that bridges the gap between CIFAR and ImageNet. Images are larger than CIFAR and there are more categories. Dataset description and files can be obtained at [43].

5. **ImageNet CLS**: There are multiple types of challenges in ImageNet, but CLS is the classification challenge, and is most commonly reported in papers. It has 1000 classes of objects with a varying amount of images per class. Most classes have 1300 examples in the training set, but a few have less than 1000. The images have variable size, typically a couple of hundred pixels wide and a couple of hundred pixels high. The images can have varying amounts of clutter and can be at different scales, making it a particularly difficult challenge. Dataset description is in [26] and the most reliable source of the data can be found at [44].

Several other classification datasets do exist but are not commonly used, such as PASCAL VOC [45] and Caltech-101 and Caltech-256 [46][3].

1.4.2 LeNet

LeNet-5 [22] is a good network to start with: it is simple yet contains many of the layers used in modern CNNs. Shown in Figure 1.9 it has two convolutional and three fully connected

[3]Tiny ImageNet is also not commonly used as it is quite new. We have included it the main list as we have found it to be quite a useful step up from CIFAR without requiring the weeks to train experimental configurations on ImageNet.

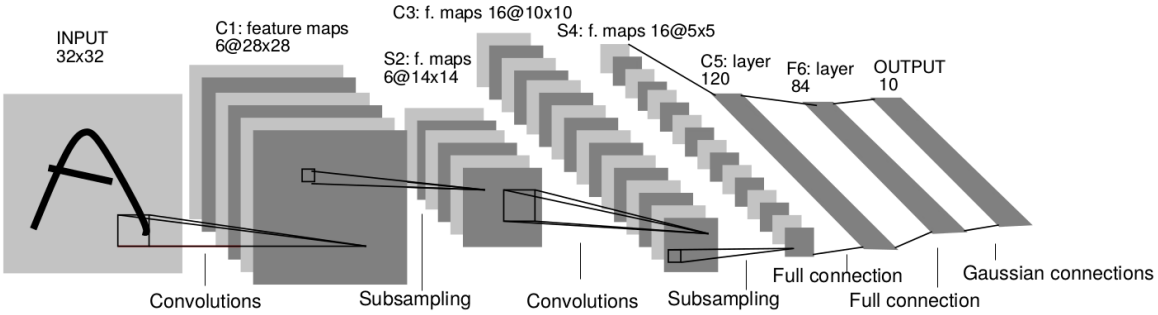

Figure 1.9: **LeNet-5 architecture.** The 'original' CNN architecture used for handwriting recognition. LeNet has 2 convolutional and 3 fully connected layers making 5 parameterized layers. After the second convolutional layer, the $16 \times 5 \times 5$ pixel output is unravelled to a 400 long vector. Image taken from [22].

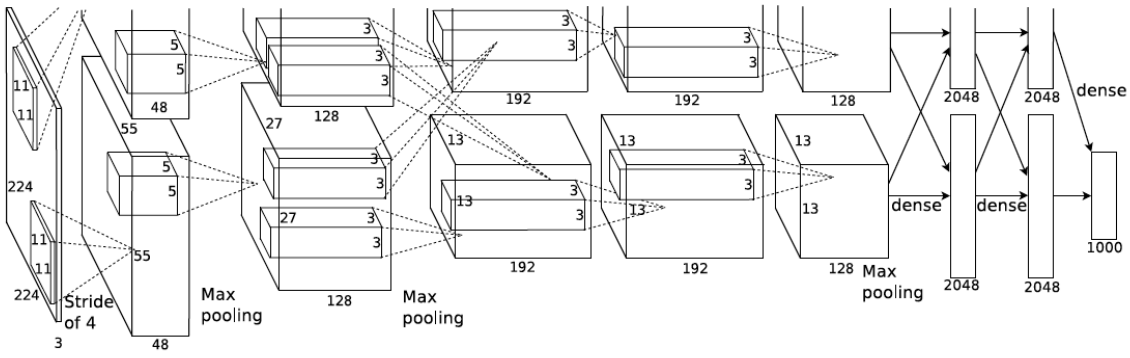

Figure 1.10: **The AlexNet architecture.** Designed for the ImageNet challenge, AlexNet may look like Figure 1.9 but is much larger. Composed of 5 convolutional layers and 3 fully connected layers. Figure taken from [25].

layers. The outputs of the convolutional layers are passed through a sigmoid nonlinearity and downsampled with average pooling. The first two fully-connected layers also have sigmoid nonlinearities. The loss function used is a combination of tanh fucntions and MSE loss.

1.4.3 AlexNet

AlexNet [25] shown in Figure 1.10 is arguably one of the most important architectures in the development in CNNs as it was able to experimentally prove that CNNs can be used for complex tasks. This required many innovations. In particular, they used multiple GPUs to do fast processing on large images, used the ReLU to avoid saturation, and added dropout to aid generalization. Training of AlexNet on 2 GPUs available in 2012 takes roughly a week.

The first layer uses convolutions with a spatial support of 11×11, followed by 5×5 and 3×3 for the final three layers.

1.4.4 VGGnet

The Visual Geometry Group (VGG) at Oxford came second in the ILSVRC challenge in 2014 with their VGG-nets [30], but remains an important network for some of the design choices it inspired. In particular, their optimal network was much deeper than AlexNet, with 19 convolutional layers on top of each other before 3 fully connected layers. These convolutional layers all used the smaller 3×3 seen only at the back of AlexNet.

This network is particularly attractive due its simplicity, compared to the more complex Inception Network [47] which won the 2014 ILSVRC challenge. VGG-16, the 16 layer variant of VGG stacks two or three convolutional layers (and ReLUs) on top of each other before reducing spatial size with max pooling. After processing at five scales, the resulting $512 \times 14 \times 14$ activation is unravelled and passed through a fully connected layer.

These VGG networks also marked the start of a trend that has since become common, where channel depth is doubled after pooling layers. The doubling of channels and quartering the spatial size still causes a net reduction in the number of activations.

1.4.5 The All Convolutional Network

The All Convolutional Network [32] introduced two popular modifications to the VGG networks:

- They argued for the removal of max pooling layers, saying that a 3×3 convolutional layer with stride 2 works just as well.
- They removed the fully connected layers at the end of the network, replacing them with 1×1 convolutions. Note that a 1×1 convolution still has shared weights across all spatial locations. The output layer then has size $C_L \times H \times W$, where H, W are many times smaller than the input image size, and the vector of C_L coefficients at each spatial location can be interpreted as a vector of scores marking the presence/absence of C_L different shapes. For classification, the output can be averaged over all spatial locations, whereas for localization it may be useful to keep this spatial information.

The new network was able to achieve state of the art results on CIFAR-10 and CIFAR-100 and competetive performance on ImageNet, while only use a fraction of the parameters of other networks.

1.4.6 Residual Networks

Resiudal Networks or ResNets won the 2015 ILSVRC challenge, introducing the residual layer. Most state of the art models today use this residual mapping in some way [34], [35].

The inspiration for the residual layer came came from the difficulties experienced in training deeper networks. Often, adding an extra layer would *decrease* network performance.

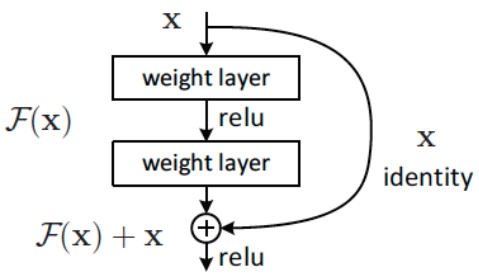

Figure 1.11: **The residual unit from ResNet.** A residual unit. The identity mapping is always present, and the network learns the difference from the identity mapping, $\mathcal{F}(x)$. Taken from [33].

This is counter-intuitive as the deeper layers could simply learn the identity mapping, and achieve the same performance.

To promote the chance of learning the identity mapping, they define a residual unit, shown in Figure 1.11. If a desired mapping is denoted $\mathcal{H}(x)$, instead of trying to learn this, they instead learn $\mathcal{F}(x) = \mathcal{H}(x) - x$. Doing this promotes a strong diagonal in the Jacobian matrix which improve conditioning for gradient descent.

Recent analysis of a ResNet without nonlinearities [48], [49] proves that SGD fails to converge for deep networks when the network mapping is far away from the identity, suggesting that a residual mapping is a good thing to do.

1.5 The Fourier and Wavelet Transforms

Computer vision is an extremely difficult task. Pixel intensities in an image are typically not very informative in understanding what is in that image. Indeed, these values are sensitive to lighting conditions and camera configurations. It would be easy to take two photos of the same scene and get two vectors x_1 and x_2 that have a very large Euclidean distance, but to a human, would represent the same objects. What is most important in defining an image is difficult to define, however some things are notably more important than others. In particular, the location or phase of the waves that make up an image is much more important than the magnitude of these waves, something that is not necessarily true for audio processing. A simple experiment to demonstrate this is shown in Figure 1.12.

1.5.1 The Fourier Transform

For a signal $f(t) \in L_2(\mathbb{R})$ (square summable signals), the *Fourier transform* is defined as:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt \tag{1.5.1}$$

Figure 1.12: **Importance of phase over magnitude for images.** The phase of the Fourier transform of the first image is combined with the magnitude of the Fourier transform of the second image and reconstructed. Note that the first image has entirely won out and nothing is left visible of the cameraman.

This can be extended to two dimensions for signals $f(\mathbf{u}) \in L_2(\mathbb{R}^2)$:

$$F(\boldsymbol{\omega}) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\mathbf{u})e^{-j\boldsymbol{\omega}^t\mathbf{u}}d\mathbf{u} = \langle f(\mathbf{u}),\ e^{j\boldsymbol{\omega}^t\mathbf{u}}\rangle \tag{1.5.2}$$

The Fourier transform is an invaluable signal expansion, as viewing a signal in the frequency space offers many insights, as well as affording many very useful properties (most notably the efficiency of convolution as a product of Fourier transforms). While it is a mainstay in signal processing, it can be a poor feature descriptor due to the infinite support of its basis functions - the complex sinusoids $e^{j\boldsymbol{\omega}^t u}$. If a single pixel changes in the input it can change all of the Fourier coefficients. As natural images are generally non-stationary, we need to be able to isolate frequency components in local regions of an image, and not have this property of global dependence.

1.5.2 The Continuous Wavelet Transform

The *Continuous Wavelet Transform* (CWT), like the Fourier Transform, can be used to decompose a signal into its frequency components. Unlike the Fourier transform, these frequency components can be localized in space. To achieve this, we need a bandpass filter, or *mother wavelet* ψ[4] such that:

$$\int_{-\infty}^{\infty} \psi(\mathbf{u})d\mathbf{u} = \Psi(0) = 0 \tag{1.5.3}$$

Any function that sufficient decay in frequency and satisfies (1.5.3) satisfies the *admissibility condition.*

[4]We use upright ψ, ϕ to distinguish 1-D wavelets from their 2-D counterparts ψ, ϕ

As we are working in 2-D for image processing, consider dilations, shifts and rotations of this function by $a > 0,\ \mathbf{b} \in \mathbb{R}^2,\ \theta \in [0, 2\pi]$ where

$$\text{Translation: } T_{\mathbf{b}}x(\mathbf{u}) = x(\mathbf{u} - \mathbf{b}) \tag{1.5.4}$$

$$\text{Dilation: } D_a x(\mathbf{u}) = \frac{1}{a} x\left(\frac{\mathbf{u}}{a}\right), \quad a > 0 \tag{1.5.5}$$

$$\text{Rotation: } R_\theta x(\mathbf{u}) = x(r_{-\theta}\mathbf{u}) \tag{1.5.6}$$

where r_θ is the 2-D rotation matrix. Now consider shifts, scales and rotations of our bandpass filter

$$\psi_{\mathbf{b},a,\theta}(\mathbf{u}) = \frac{1}{a}\psi\left(\frac{r_{-\theta}(\mathbf{u} - \mathbf{b})}{a}\right) \tag{1.5.7}$$

which are called the *daughter wavelets*. The 2D CWT of a signal $x(\mathbf{u})$ is defined as

$$CWT_x(\mathbf{b}, a, \theta) = \int_{-\infty}^{\infty} \psi^*_{\mathbf{b},a,\theta}(\mathbf{u}) x(\mathbf{u}) d\mathbf{u} = \langle \psi_{\mathbf{b},a,\theta}(\mathbf{u}),\ x(\mathbf{u}) \rangle \tag{1.5.8}$$

1.5.2.1 Properties

The CWT has some particularly nice properties. In particular, it has *covariance* under the three transformations (1.5.4)-(1.5.6):

$$T_{\mathbf{b}_0} x \to CWT_x(\mathbf{b} - \mathbf{b}_0, a, \theta) \tag{1.5.9}$$

$$D_{a_0} x \to CWT_x(\mathbf{b}/a_0, a/a_0, \theta) \tag{1.5.10}$$

$$R_{\theta_0} x \to CWT_x(r_{-\theta_0}\mathbf{b}, a, \theta + \theta_0) \tag{1.5.11}$$

Most importantly, the CWT is now localized in space, which distinguishes it from the Fourier transform. This means that changes in one part of the image will not affect the wavelet coefficients in another part of the image, so long as the distance between the two parts is much larger than the wavelength of the wavelets you are examining.

1.5.2.2 Inverse

The CWT can be inverted by using a *dual* function $\tilde{\psi}$. There are restrictions on what dual function we can use, namely the dual-wavelet pair must have an admissible constant C_ψ that satisfies the cross-admissibility constraint [50]. Assuming these constraints are satisifed, we can recover x from CWT_x by:

$$x(\mathbf{u}) = \frac{1}{C_\psi} \int\int\int \frac{1}{a^3} CWT_x(\mathbf{b}, a, \theta) \tilde{\psi}_{\mathbf{b},a,\theta}\ d\mathbf{b} da d\theta \tag{1.5.12}$$

1.5.2.3 Interpretation

As the CWT is a convolution with a zero mean function, the wavelet coefficients are only large in the regions of the parameter space $(\mathbf{b}, a, \theta)$ where $\psi_{\mathbf{b},a,\theta}$ 'match' the features of the signal. As the wavelet ψ is well localized, the energy of the coefficients CWT_x will be concentrated on the significant parts of the signal.

For an excellent description of the properties of the CWT in 1-D we recommend [51] and in 2-D we recommend [52].

1.5.3 Discretization and Frames

The CWT is highly redundant. We have taken a 2-D signal and expressed it in 4 dimensions (2 offset, 1 scale and 1 rotation). In reality, we would like to sample the space of the CWT. We would ideally like to fully retain all information in x (be able to reconstruct x from the samples) while sampling over $(\mathbf{b}, a, \theta)$ as little as possible to avoid redundancy. To understand how to do this we must briefly talk about frames.

A set of vectors $\phi = \{\varphi_i\}_{i \in I}$ in a hilbert space $\mathbb{H}$ is a *frame* if there exist two constants $0 < A \leq B < \infty$ such that for all $x \in \mathbb{H}$:

$$A||x||^2 \leq \sum_{i \in I} |\langle x, \varphi_i \rangle|^2 \leq B||x||^2 \tag{1.5.13}$$

with A, B called the *frame bounds* [53]. The frame bounds relate to the issue of stable reconstruction. In particular, no vector x with $||x|| > 0$ should be mapped to 0, as this would violate the bound on A from below. This can be interpreted as ensuring our set ϕ covers the entire frequency space. The upper bound ensures that the transform coefficients are bounded.

Any finite set of vectors that spans the space is frame. An orthogonal basis is a commonly known frame where $A = B = 1$ and $|\varphi_i| = 1$ (e.g. the Discrete Wavelet Transform or the Fourier Transform). Tight frames are frames where $A = B$ and Parseval tight frames have the special case $A = B = 1$. It is possible to have frames that have more vectors than dimensions, and this will be the case with many expansions we explore in this thesis.

If $A = B$ and $|\varphi_i| = 1$, then A is the measure of the redundancy of the frame. Of course, for the orthogonal basis, $A = 1$ when $|\varphi_i| = 1$ so there is no redundancy. For the 2-D DT$\mathbb{C}$WT which we will see shortly, the redundancy is 4.

1.5.3.1 Inversion and Tightness

(1.5.13) specify the constraints that make a frame representation invertible. The tighter the frame bounds, the more easily it is to invert the signal. Tight representations have other particular benefits **finish me**. This gives us some guide to choosing the sampling grid for the CWT.

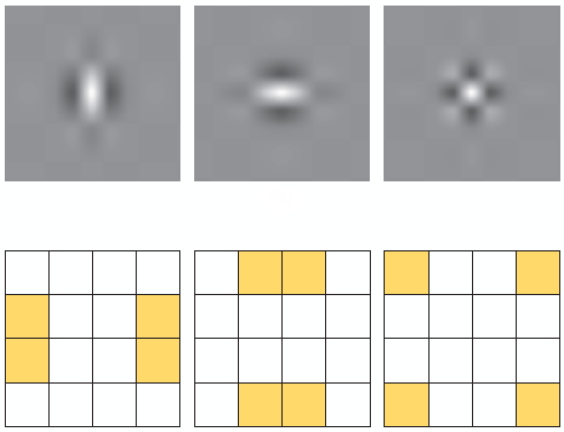

Figure 1.13: **Typical wavelets from the 2D separable DWT.** Top: Wavelet point spread functions for ψ^v (low-high), ψ^h (high-low), and ψ^d (high-high) wavelets. High-high wavelets are in a checkerboard pattern, with no favoured orientation. Bottom: Idealized support of the spectra of each of the wavelets. Image taken from [54].

One particular inverse operator is the *canonical dual frame*. If we define the frame operator $S = \Phi\Phi^*$ then the canonical dual of Φ is defined as $\tilde{\Phi} = \tilde{\varphi}_{i\in I}$ where:

$$\tilde{\varphi}_i = S^{-1}\varphi_i \tag{1.5.14}$$

then[53]

$$x = \sum_{i\in I}\langle x, \varphi_i\rangle\tilde{\varphi}_i = \sum_{i\in I}\langle x, \tilde{\varphi}_i\rangle\varphi_i \tag{1.5.15}$$

If a frame is tight, then so is its dual.

1.5.4 Discrete Wavelet Transform

(1.5.7) gave the equation for the daughter wavelets in 2-D, in 1-D at scales $a = 2^j, j \geq 0$, this is simply:

$$\Psi_{b,j}(u) = 2^{-j/2}\Psi\left(\frac{u-b}{2^j}\right) \tag{1.5.16}$$

The 2-D DWT has one scaling fucntion and three wavelet functions, composed of the product of 1-D wavelets in the horizontal and vertical directions:

$$\phi(\mathbf{u}) = \Phi(u_1)\Phi(u_2) \tag{1.5.17}$$
$$\psi^h(\mathbf{u}) = \Phi(u_1)\Psi(u_2) \tag{1.5.18}$$
$$\psi^v(\mathbf{u}) = \Psi(u_1)\Phi(u_2) \tag{1.5.19}$$
$$\psi^d(\mathbf{u}) = \Psi(u_1)\Psi(u_2) \tag{1.5.20}$$

with h, v, d indicating the sensitivity to horizontal, vertical and diagonal edges. The point spread functions for the wavelet functions are shown in Figure 1.13.

For the four equations above (1.5.17) – (1.5.20), define the daughter wavelets as:

$$\phi_{kl}^{j}(\mathbf{u}) = \phi_{j,k}(u_1)\phi_{j,l}(u_2) \tag{1.5.21}$$
$$\psi_{kl}^{h,j}(\mathbf{u}) = \phi_{j,k}(u_1)\psi_{j,l}(u_2) \tag{1.5.22}$$
$$\psi_{kl}^{v,j}(\mathbf{u}) = \psi_{j,k}(u_1)\phi_{j,l}(u_2) \tag{1.5.23}$$
$$\psi_{kl}^{d,j}(\mathbf{u}) = \psi_{j,k}(u_1)\psi_{j,l}(u_2) \tag{1.5.24}$$

for $\alpha = \{h, v, d\},\ k, l \in \mathbb{Z}$. We can then get an orthonormal basis with the set $\{\phi_{kl}^{j}, \psi_{kl}^{\alpha,j}\}$. The wavelet coefficients at chosen scale and location can then be found by taking the inner product of the signal x with the daughter wavelets.

1.5.4.1 Shortcomings

The Discrete Wavelet Transform (DWT) is an orthogonal basis. It is a natural first signal expansion to consider when frustrated with the limitations of the Fourier Transform. It is also a good example of the limitations of non-redundant transforms, as it suffers from several drawbacks:

- The DWT is sensitive to the zero crossings of its wavelets. We would like singularities in the input to yield large wavelet coefficients, but if they fall at a zero crossing of a wavelet, the output can be small. See Figure 1.14.
- They have poor directional selectivity. As the wavelets are purely real, they have passbands in all four quadrants of the frequency plane. While they can pick out edges aligned with the frequency axis, they do not have admissibility for other orientations. See Figure 1.13.
- They are not shift invariant. In particular, small shifts greatly perturb the wavelet coefficients. Figure 1.14 shows this for the centre-left and centre-right images.

The lack of shift invariance and the possibility of low outputs at singularities is a price to pay for the critically sampled property of the transform. This shortcoming can be overcome with the undecimated DWT [55], [56], but it comes with a heavy computational and memory cost.

1.5.5 Complex Wavelets

Fortunately, we can improve on the DWT with complex wavelets, as they can solve these new shortcomings while maintaining the desired localization properties.

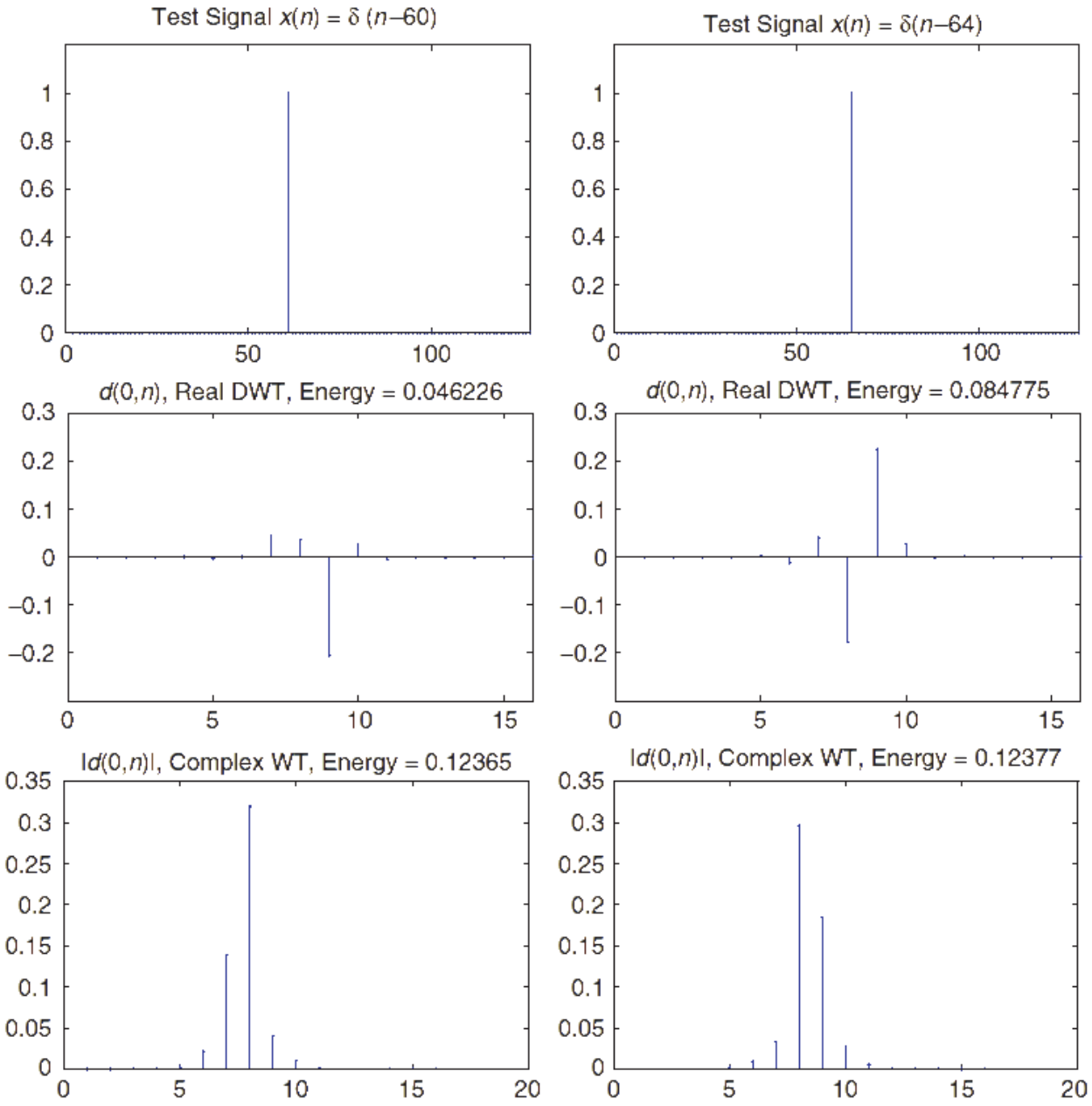

Figure 1.14: **Sensitivity of DWT coefficients to zero crossings and small shifts.** Two impulse signals $\delta(n-60)$ and $\delta(n-64)$ are shown (top), as well as the wavelet coefficients for scale $j=1$ for the DWT (middle) and for the DT$\mathbb{C}$WT (bottom). In the middle row, not only are the coefficients very different from a shifted input, but the energy has almost doubled. As the DWT is an orthonormal transform, this means that this extra energy has come from other scales. In comparison, the energy of the magnitude of the DT$\mathbb{C}$WT coefficients has remained far more constant, as has the shape of the envelope of the output. Image taken from [54].

The Fourier transform does not suffer from a lack of directional selectivity and shift variance, because its basis functions are based on the complex sinusoid:

$$e^{j\omega t} = \cos(\omega t) + j\sin(\omega t) \tag{1.5.25}$$

whereas the DWT's basis functions are based on only the real sinusoid $\cos(\omega t)$.[5] As t moves along the real line, the phase of the Fourier coefficients change linearly, while their magnitude remains constant. In contrast, as t moves along the real line, the sign of the real coefficient flips between -1 and 1, and its magnitude is a rectified sinusoid.

These nice properties come from the fact that the cosine and sine functions of the Fourier transform form a Hilbert Pair and together constitute an analytic signal.

We can achieve these nice properties if the mother wavelet for our wavelet transform is analytic:

$$\psi_c(t) = \psi_r(t) + j\psi_i(t) \tag{1.5.26}$$

where $\psi_r(t)$ and $\psi_i(t)$ form a Hilbert Pair (i.e., they are 90° out of phase with each other).

There are a number of possible ways to do a wavelet transform with complex wavelets. We examine two in particular, a Fourier-based, sampled CWT using Morlet wavelets, and the Dual-Tree Complex Wavelet Transform (DT$\mathbb{C}$WT) developed by Kinsbury [54], [57]–[63].

The Morlet wavelet transform we look at as it is used by Mallat et. al. in their scattering transform [64]–[72], which has been a large inspiration for our work, and we will introduce it shortly. The DT$\mathbb{C}$WT we believe offers several advantages over the Morlet based implementation, and has been the basis for most of our work.

Let us write the wavelet transform of an input x with as

$$\mathcal{W}x = \{x * \phi_J, x * \psi_\lambda\}_\lambda \tag{1.5.27}$$

where $\lambda = (j,k)$ indexes the J scales and K orientations of the chosen wavelet transform, whether it be the DT$\mathbb{C}$WT or Morlet transform.

1.5.6 Sampled Morlet Wavelets

The wavelet transform used by Mallat et. al. in their scattering transform are an efficient implementation of the Gabor Transform. While the Gabor wavelets have the best theoretical trade-off between spatial and frequency localization, they have a non-zero mean. This violates (1.5.3) making them inadmissible as wavelets. Instead, the Morlet wavelet has the same shape, but with an extra degree of freedom chosen to set $\int \psi(\mathbf{u})d\mathbf{u} = 0$. This wavelet has

[5] we have temporarily switched to 1D notation here as it is clearer and easier to use, but the results still hold for 2D

Figure 1.15: **Single Morlet filter with varying slants and window sizes.** Top left — 45° plane wave (real part only). Top right — plane wave with $\sigma = 3, \gamma = 1$. Bottom left — plane wave with $\sigma = 3, \gamma = 0.5$. Bottom right — plane wave with $\sigma = 2, \gamma = 0.5$.

equation (in 2D):

$$\psi(\mathbf{u}) = \frac{1}{2\pi\sigma^2}(e^{i\mathbf{u}\xi} - \beta)e^{-\frac{|\mathbf{u}|^2}{2\sigma^2}} \tag{1.5.28}$$

where β is usually $<< 1$ and is this extra degree of freedom, σ is the size of the gaussian window, and ξ is the location of the peak frequency response — i.e., for an octave based transform, $\xi = 3\pi/4$.

Bruna and Mallat add an extra degree of freedom in their original design [65] by allowing for a non-circular Gaussian window over the complex sinusoid, which gives control over the angular resolution of the final wavelet. (1.5.28) now becomes:

$$\psi(\mathbf{u}) = \frac{\gamma}{2\pi\sigma^2}(e^{i\mathbf{u}\xi} - \beta)e^{-\mathbf{u}^\mathbf{t}\Sigma^{-1}\mathbf{u}} \tag{1.5.29}$$

Where

$$\Sigma^{-1} = \begin{bmatrix} \frac{1}{2\sigma^2} & 0 \\ 0 & \frac{\gamma^2}{2\sigma^2} \end{bmatrix}$$

The effects of modifying the eccentricity parameter γ and the window size σ are shown in Figure 1.15. A full family of Morlet wavelets at varying scales and orientations is shown in **??**.

1.5.6.1 Tightness and Invertibility

Recall our definition of the wavelet transform $\mathcal{W}$ from (1.5.27).

Assuming the transform is bounded, we can always scale it so that it satisfies Plancherel's equality

$$\|\mathcal{W}x\| = \|x\| \tag{1.5.30}$$

which is a nice property to have for invertibility, as well as for analysing how different signals get transformed (e.g. white noise versus standard images). Scaling the transform changes the upper bound B in (1.5.13) to 1 and makes the lower bound $A = 1 - \alpha$, where α is a measure of how non-tight a frame is.

Let us look at the tightness of a Morlet wavelet frame for a few manually selected parameters.

- For dilations, we choose $a = 2^{-j/Q}$ for $j \in \mathbb{Z}$ controlling the scale and Q the number of octaves per scale.
- For rotations, we subdivide the interval $[0, \pi)$ into K sections, and choose $\theta_k = \frac{k\pi}{K}$, $k = \{0, 1, \ldots K-1\}$.
- For the translations, we set the sample spacing $\Delta \mathbf{b} = 2^{-j/Q}$. **Need to check this**

Using the capital notation to denote the Fourier transform, define the function $A(\boldsymbol{\omega})$ to be the coverage each wavelet family has over the frequency plane:

$$A(\boldsymbol{\omega}) = |\Phi_J(\boldsymbol{\omega})|^2 + \sum_{\lambda} |\Psi_\lambda(\boldsymbol{\omega})|^2 \tag{1.5.31}$$

For a unit norm input $||x||^2 = 1$ and scaled wavelets, we can now change (1.5.13) to be:

$$1 - \alpha \leq A(\boldsymbol{\omega}) \leq 1 \tag{1.5.32}$$

If $A(\boldsymbol{\omega})$ is ever close to 0, then there is not a good coverage of the frequency plane at that location. If it ever exceeds 1, then there is overlap between bases. Both of these conditions make invertibility difficult[6]. Figure 1.16 show the frequency coverage of a few sample grids over the CWT parameters used by Mallat et. al.. Invertibility is possible, but not guaranteed for all configurations.

1.5.7 The DT$\mathbb{C}$WT

The DT$\mathbb{C}$WT was first proposed by Kingsbury in [58], [59] as a way to combat many of the shortcomings of the DWT, in particular, its poor directional selectivity, and its poor shift invariance. A thorough analsysis of the properties and benefits of the DT$\mathbb{C}$WT is done in [54], [60]. Building on these properties, it been used successfully for denoising and inverse problems [73]–[76], texture classification [77], [78], image registration [79], [80] and SIFT-style keypoint generation matching [81]–[85] amongst many other applications. Compared to Gabor (or Morlet) image analysis, the authors of [54] sum up the dangers as:

[6] In practise, if $A(\boldsymbol{\omega})$ is only slightly greater 1 for only a few small areas of $\boldsymbol{\omega}$, approximate inversion can be achieved

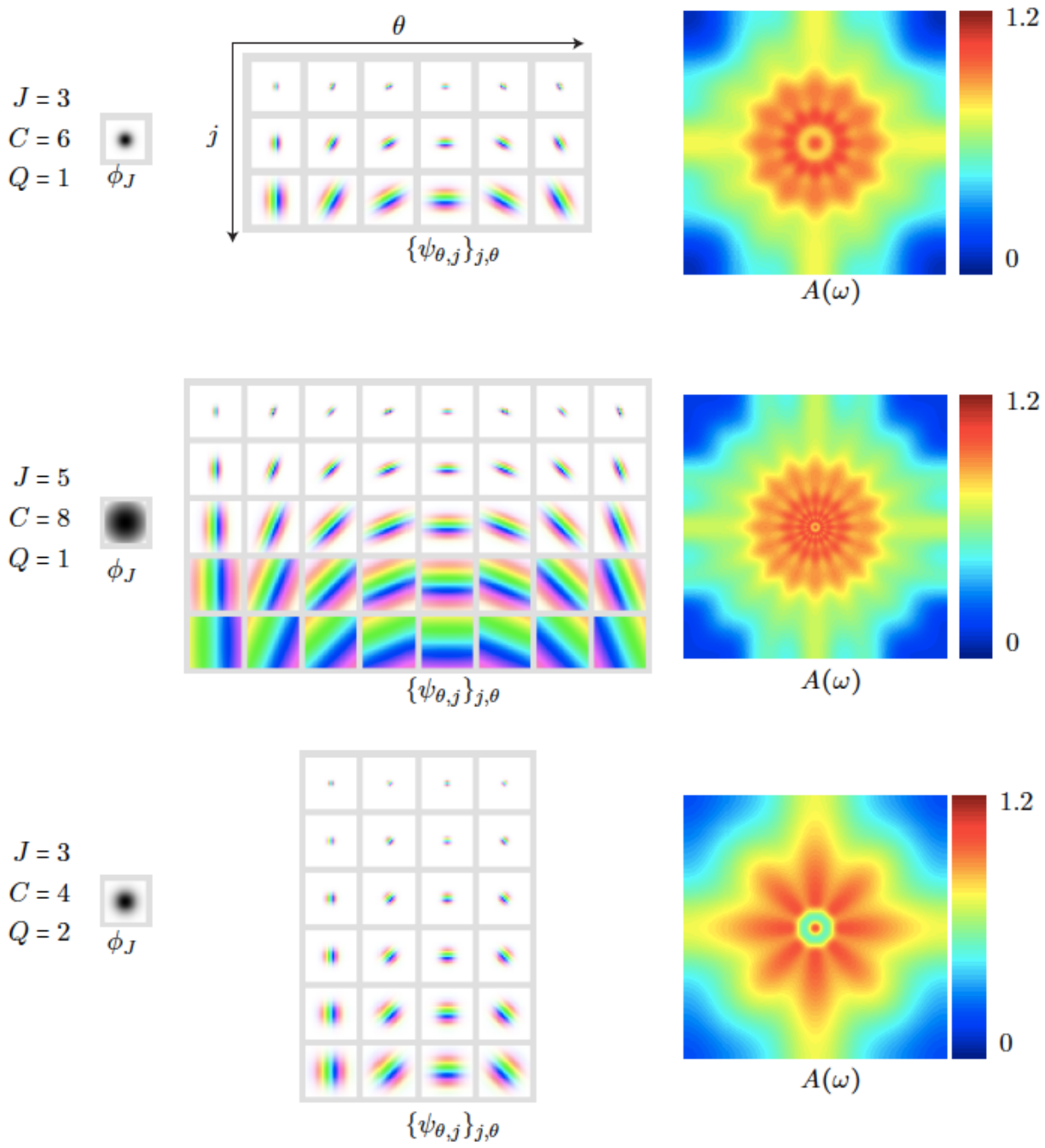

Figure 1.16: **Three Morlet Wavelet families and their tiling of the frequency plane.** For each set of parameters, the point spread functions of the wavelet bases are shown, next to their covering of the frequency plane $A(\boldsymbol{\omega})$. None of the configurations cover the corners of the frequency plane, but this is often mostly noise. Increasing J, K (Sifre uses C in these diagrams) or Q gives better frequency localization but at the cost of spatial localization and added complexity. Image taken from [71]. **TODO: update figure to show only the real wavelets in black and white and the correct variable names**.

> A typical Gabor image analysis is either expensive to compute, is noninvertible, or both.

This nicely summarises the difference between this method and the Fourier based method outlined in subsection 1.5.6. The DT$\mathbb{C}$WT is a filter bank (FB) based wavelet transform. It is faster to implement than the Morlet analysis, as well as being more readily invertible.

1.5.7.1 Deisgn Criteria for the DT$\mathbb{C}$WT

As in subsection 1.5.5, we want to have a complex mother wavelet $\psi_c = \psi_r + j\psi_i$ and complex father wavelet $\phi_c = \phi_r + j\phi_i$, but now achieved with filter banks. The complex componenet allows for support of both the mother and father wavelet on only one half of the frequency plane.

The dual tree framework shown in Figure 1.17 can achieve this by making the real and imaginary components with their own DWT. In particular, if we define:

- h_0, h_1 the low and high-pass analysis filters for ϕ_r, ψ_r
- g_0, g_1 the low and high-pass analysis fitlers for ϕ_i, ψ_i
- $\tilde{h}_0, \tilde{h}_1$ the low and high pass synthesis filters for $\tilde{\phi}_r, \tilde{\psi}_r$.
- $\tilde{g}_0, \tilde{g}_1$ the low and high pass synthesis filters for $\tilde{\phi}_i, \tilde{\psi}_i$.

The dilation and wavelet equations for a 1D filter bank implementation are:

$$\phi_r(t) = \sqrt{2}\sum_n h_0(n)\phi_r(2t-n) \tag{1.5.33}$$

$$\psi_r(t) = \sqrt{2}\sum_n h_1(n)\phi_r(2t-n) \tag{1.5.34}$$

$$\phi_i(t) = \sqrt{2}\sum_n g_0(n)\phi_i(2t-n) \tag{1.5.35}$$

$$\psi_i(t) = \sqrt{2}\sum_n g_1(n)\phi_i(2t-n) \tag{1.5.36}$$

Designing a filter bank implementation that results in Hilbert symmetric wavelets does not appear to be an easy task. However, it was shown by Kingsbury in [60] (and later proved by Selesnick in [86]) that the necessary conditions are conceptually very simple. One low-pass filter must be a *half-sample shift* of the other. I.e.,

$$g_0(n) \approx h_0(n-0.5) \rightarrow \psi_g(t) \approx \mathcal{H}\{\psi_h(t)\} \tag{1.5.37}$$

As the DT$\mathbb{C}$WT is designed as an invertible filter bank implementation, this is only one of the constraints. Naturally, there are also perfect reconstruction, finite support, linear phase and vanishing moment constraints to consider in the filter bank design.

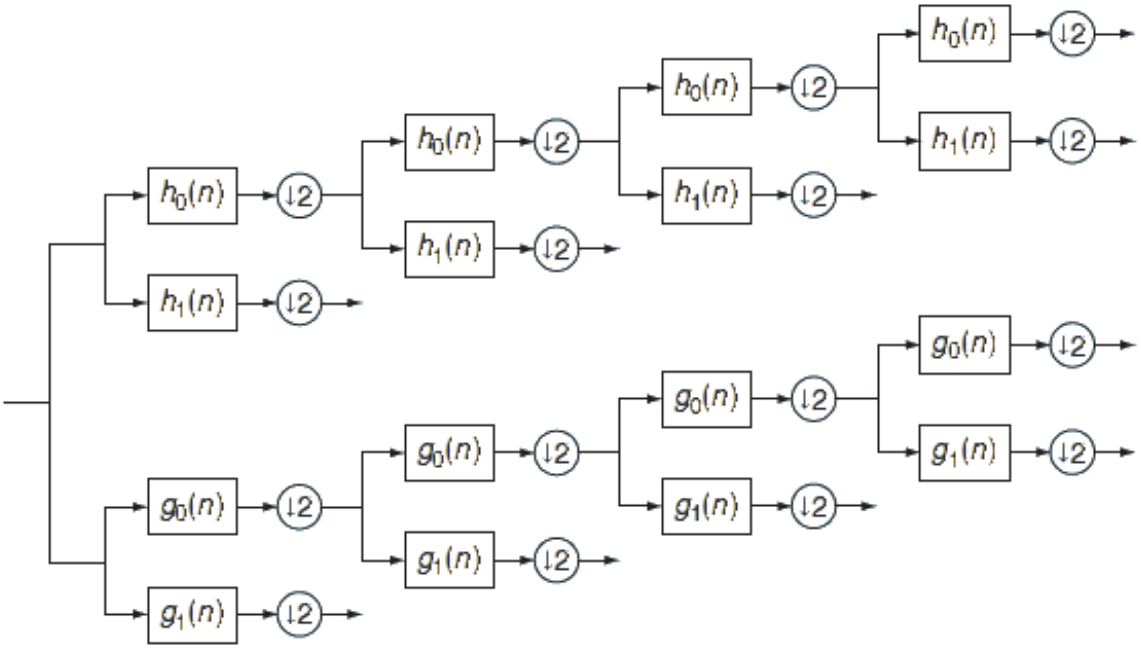

Figure 1.17: **Analysis FB for the** DT$\mathbb{C}$WT**.** Top 'tree' forms the real component of the complex wavelet ψ_r, and the bottom tree forms the imaginary (Hilbert pair) component ψ_i. Image taken from [54].

The derivation of the filters that meet these conditions is covered in detail in [63], [87], and in general in [54]. The result is the option of three families of filters: biorthogonal filters ($h_0[n] = h_0[N-1-n]$ and $g_0[n] = g_0[N-n]$), q-shift filters ($g_0[n] = h_0[N-1-n]$), and common-factor filters.

1.5.7.2 2-D DT$\mathbb{C}$WT and its Properties

While analytic wavelets in 1D are useful for their shift invariance, the real beauty of the DT$\mathbb{C}$WT is in its ability to make a separable 2D wavelet transform with oriented wavelets.

Figure 1.18a shows the spectrum of the wavelet when the separable product uses purely real wavelets, as is the case with the DWT. Figure 1.18b however, shows the separable product of two complex, analytic wavelets resulting in a localized and oriented 2D wavelet.

Note that in this thesis, we name the wavelets by the direction of the edge that they are most sensitive to.

For example, the 135° wavelet can be obtained by the separable product:

$$\psi(\mathbf{u}) = \psi_c(u_1)\psi_c^*(u_2) \tag{1.5.38}$$

$$= (\psi_r(u_1) + j\psi_i(u_1))\,(\psi_r(u_2) - j\psi_i(u_2)) \tag{1.5.39}$$

$$= (\psi_r(u_1)\psi_r(u_2) - \psi_i(u_1)\psi_i(u_2)) + j\,(\psi_r(u_1)\psi_i(u_2) + \psi_i(u_1)\psi_r(u_2)) \tag{1.5.40}$$

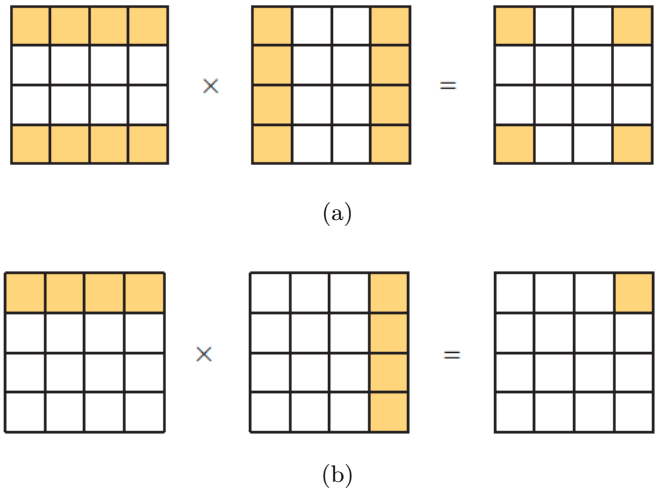

Figure 1.18: **The DWT high-high vs the** DT$\mathbb{C}$WT **high-high frequency support.** (a) The high-high DWT wavelet having a passband in all 4 corners of the frequency plane vs (b) the high-high DT$\mathbb{C}$WT wavelet frequency support only existing in one quadrant. Taken from [54]

Similar equations can be obtained for the other five wavelets and the scaling function, by replacing ψ with ϕ for both directions, and not taking the complex conjugate in (1.5.38) to get the right hand side of the frequency plane. The 2-D DT$\mathbb{C}$WT requires four 2-D DWTs to calculate the four possible combinations of real and imaginary components. The high and lowpass outputs from these DWTs can then be summed in different ways as in (1.5.40) to get the complex bandpass wavelets. Figure 1.19 shows the resulting wavelets both in the spatial domain and their idealized support in the frequency domain.

1.5.7.3 Tightness and Invertibility

We analysed the coverage of the frequency plane for the Morlet wavelet family and saw what areas of the spectrum were better covered than others. How about for the DT$\mathbb{C}$WT?

It is important to note that in the case of the DT$\mathbb{C}$WT, the wavelet transform is also approximately unitary, i.e.,

$$\|x\|^2 \approx \|\mathcal{W}x\|^2 \tag{1.5.41}$$

and the implementation is perfectly invertible as $A(\boldsymbol{\omega})$ from (1.5.31) function is unity (or very near unity) $\forall \boldsymbol{\omega} \in [-\pi, \pi] \times [-\pi, pi]$. See Figure 1.20. This is not a surprise, as it is a design constraint in choosing the filters, but nonetheless is important to note.

1.5.8 Summary of Methods

One final comparison to make between the DT$\mathbb{C}$WT and the Morlet wavelets is their frequency coverage. The Morlet wavelets have flexibility at the cost of computational expense, and can

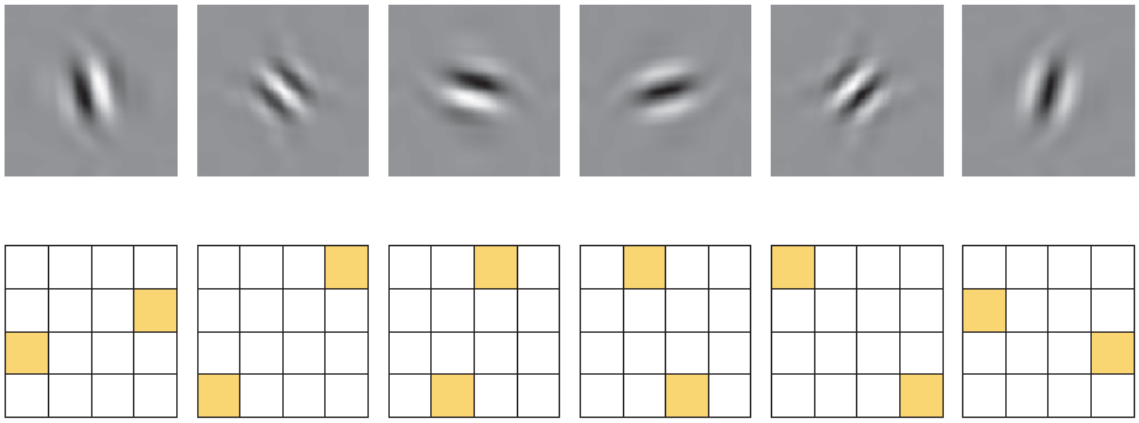

Figure 1.19: **Wavelets from the 2d** DT$\mathbb{C}$WT**. Top:** The six oriented filters in the space domain (only the real wavelets are shown). From left to right these are the $105°, 135°, 165°, 15°, 45°, 75°$ wavelets. **Bottom:** Idealized support of the Fourier spectrum of each wavelet in the 2D frequency plane. Spectra of the the real wavelets are shown — the spectra of the complex wavelets $(\psi_r + j\psi_i)$ only has support in the top half of the plane. Image taken from [54].

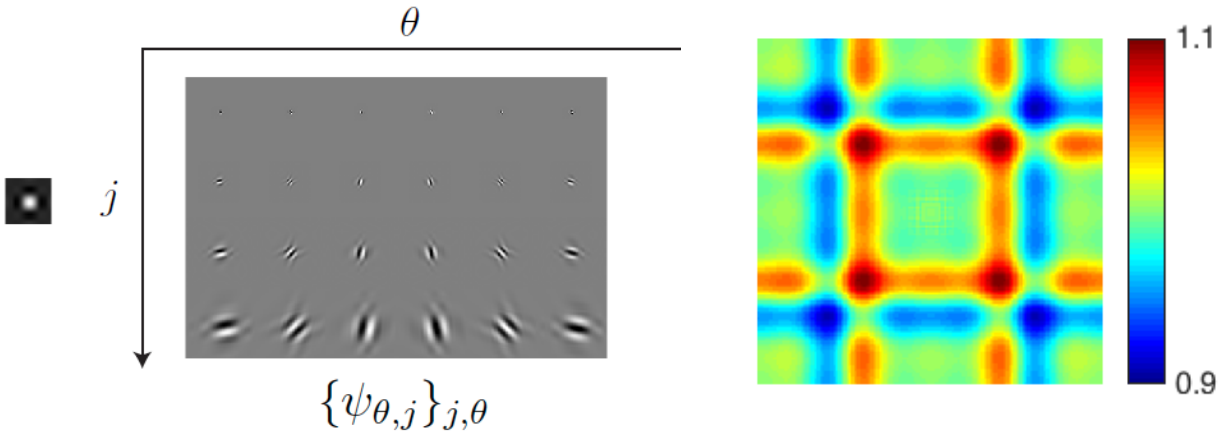

Figure 1.20: DT$\mathbb{C}$WT **family for** $J = 4$ **and their frequency coverage.** Note the reduced scale compared to Figure 1.16.

be made to have tighter angular resolution than the DT$\mathbb{C}$WT. However it is not always better to keep using finer and finer resolutions, indeed the Fourier transform gives the ultimate in angular resolution, but as mentioned, this makes it less stable to shifts and deformations. We will explore this in more depth in Chapter 3.

1.6 Scatternets

Scatternets have been a very large influence on our work, as well as being quite distinct from the previous discussions on learned methods. They were first introduced by Bruna and Mallat in their work [64], and then were rigorously defined by Mallat in [88].

While CNNs have the ability to learn invariances to nuiscance variabilities, the properties and optimal configurations are not well understood. It typically takes multiple trials by an expert to find the correct hyperparameters for these networks. A scattering transform instead builds well understood and well defined invariances.

We first review some of the desirable invariances before describing how a ScatterNet achieves them.

1.6.1 Desirable Properties

1.6.1.1 Translation Invariance

Translation is often defined as being uninformative for classification — an object appearing in the centre of the image should be treated the same way as an the same object appearing in the corner of an image, i.e., a representation Φx is invariant to global translations $x_c(\mathbf{u}) = x(\mathbf{u} - \mathbf{c})$ by $\mathbf{c} = (c_1, c_2) \in \mathbb{R}^2$ if

$$\|\Phi x_c - \Phi x\| \leq C \tag{1.6.1}$$

for some small constant $C > 0$. Note that we may instead want only local translation invariance and restrict the distance $|\mathbf{c}|$ for which (1.6.1) is true.

Note that CNNs are naturally covariant to translations in the pixel space, so $\Phi x_c = (\Phi x)_c$, $\mathbf{c} \in \mathbb{Z}^2$. Of course, natural objects exist in continuous space and are sampled, and any two images of the same scene taking with small camera disturbances are unlikely to be at integer pixel shifts of each other.

1.6.1.2 Stability to Noise

Stability to additive noise is another useful invariance to build, as it is a common feature in sampled signals. Stability is defined in terms of Lipschitz continuity, which is a strong form of uniform continuity for functions, which we briefly introduce here.

Formally, a Lipschitz continuous function is limited in how fast it can change; there exists an upper bound on the gradient the function can take, although it doesn't necessarily need to

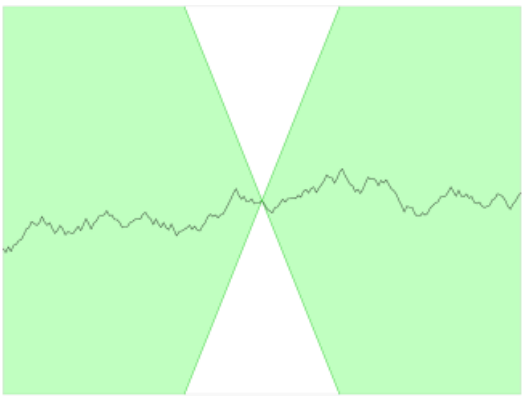

Figure 1.21: **A Lipschitz continuous function.** There is a cone for this function (shown in white) such that the graph always remains entirely outside the cone as it is shifted across. The minimum gradient needed for this to hold is called the 'best Lipschitz constant'.

be differentiable everywhere. The modulus operator $|x|$ is a good example of a function that has a bounded derivative and so is Lipschitz continuous, but isn't differentiable everywhere. Alternatively, the modulus squared has derivative everywhere but is not Lipshitz continuous as its gradient grows with x.

To be stable to additive noise, we require that for a new signal $x'(\mathbf{u}) = x(\mathbf{u}) + \epsilon(\mathbf{u})$, there must exist a bounded $C > 0$ s.t.

$$\|\Phi x' - \Phi x\| \leq C\|x' - x\| \tag{1.6.2}$$

1.6.1.3 Stability to Deformations

Small deformations are important to be invariant to. However, this must be limited. It is important to ignore intra-class variations but not so invariant that an object can morph into another (in the case of MNIST for example, we do not want to be so stable to deformations that 7s can map to 1s).

Formally, for a new signal $x_\tau(\mathbf{u}) = x(\mathbf{u} - \tau(\mathbf{u}))$, where $\tau(\mathbf{u})$ is a non constant displacement field (i.e., not just a translation) that deforms the image, we require a $C_\tau > 0$ s.t.

$$\|\Phi x_\tau - \Phi x\| \leq C_\tau \|x\| \sup_{\mathbf{u}} |\nabla \tau(\mathbf{u})| \tag{1.6.3}$$

The term on the right $|\nabla \tau(\mathbf{u})|$ measures the deformation amplitude, so the supremum of it is a limit on the global defomation amplitude.

1.6.2 Definition

A Fourier modulus satisfies the first two of these requirements, in that it is both translation invariant and stable to additive noise, but it is unstable to deformations due to the infinite support of the sinusoid basis functions it uses. It also loses too much information — very different signals can all have the same Fourier modulus, e.g. a chirp, white noise and the Dirac delta function all have flat spectra.

Another translation invariant and stable operator is the averaging kernel, and Mallat et. al. use this to make the zeroth scattering coefficient:

$$S[\emptyset]x \triangleq x * \phi_J \left(2^J \mathbf{u}\right) \tag{1.6.4}$$

which is translation invariant to shifts less than 2^J. It unfortunately results in a loss of information due to the removal of high frequency content. This is easy to see as the wavelet operator $Wx = \{x*\phi_J, x*\psi_\lambda\}_\lambda$ contains all the information of x, whereas the zeroth scattering coefficient is simply the lowpass portion of W.

This high frequency content can be 'recovered' by keeping the wavelet coefficients. The wavelet terms, like a convolutional layer in a CNN, is only covariant to shifts rather than invariant. This covariance happens in the real and imaginary parts which both vary rapidly. Fortunately, its modulus is much smoother and gives a good measure for the frequency-localized energy content at a given spatial location[7]. Unlike the Fourier modulus, the complex wavelet modulus is stable to deformations due to the grouping together frequencies into dyadic packets [88].

We combine the wavelet transform and modulus operators into one operator $\tilde{W}$:

$$\tilde{W}x = \{x*\phi_J,\ |x*\psi_\lambda|\}_\lambda \tag{1.6.5}$$

$$= \{x*\phi_J,\ U[\lambda]x\}_\lambda \tag{1.6.6}$$

where the U terms are called the *propagated* signals and $\lambda = (j,k)$ indexes the scale and orientation of wavelet used. These U terms are invariant for shifts of up to 2^j. Mallat et. al. choose to keep the same level of invariance as the zeroth order coefficients (2^J) by further averaging. This makes the first ordering scattering coefficients:

$$S[\lambda_1]x \triangleq U[\lambda_1]x * \phi_J = |x*\psi_{\lambda_1}|*\phi_J \tag{1.6.7}$$

Again this averaging comes at a cost of discarding high frequency information, this time about the wavelet sparsity signal $U[\lambda] = |x*\psi_\lambda|$ instead of the input signal x. We can recover

[7] Interestingly, the modulus operator can often still be inverted, and hence does not lose any information, due to the redundancies of the complex wavelet transform [89]

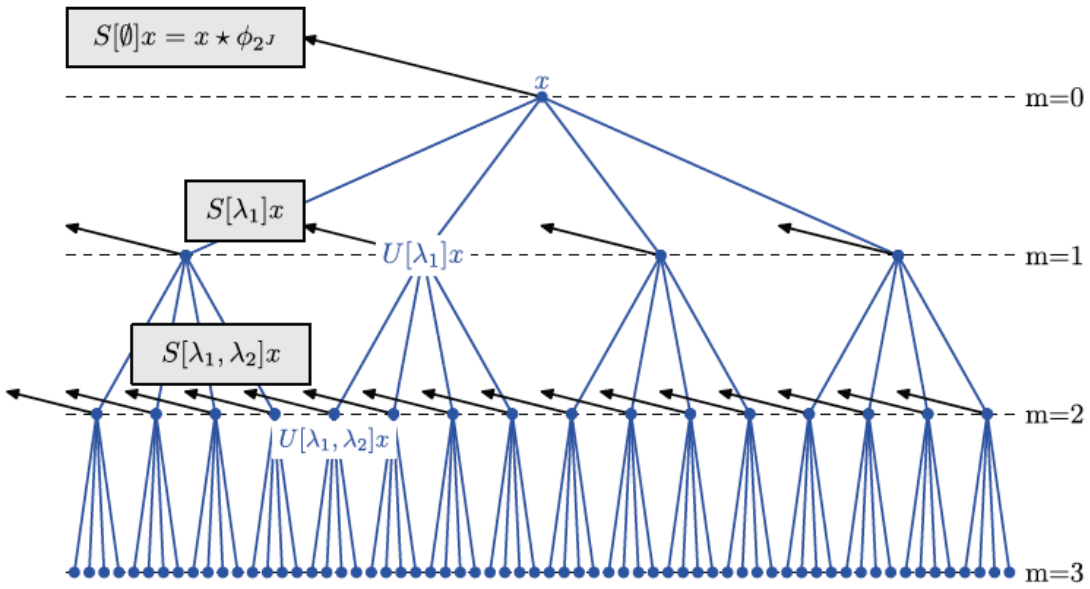

Figure 1.22: **The Scattering Transform.** Scattering outputs are the leftward pointing arrows $S[p]x$, and the intermediate coefficients $U[p]x$ are the centre nodes of the tree. Taken from [65].

this information by repeating the above process.

$$S[\lambda_1, \lambda_2]x \triangleq U[\lambda_2]U[\lambda_1]x \tag{1.6.8}$$

$$= ||x * \psi_{\lambda_1}| * \psi_{\lambda_2}| * \phi_J \tag{1.6.9}$$

In general, let $p = (\lambda_1, \lambda_2, \ldots \lambda_m)$ be a path of length m describing the order of application of wavelets, and define:

$$U[p]x = U[\lambda_m]U[\lambda_{m-1}] \cdots U[\lambda_1]x \tag{1.6.10}$$

$$= ||\cdots|x * \psi_{\lambda_1}| * \psi_{\lambda_2}| \cdots * \psi_{\lambda_m}| \tag{1.6.11}$$

and the mth order scattering coefficient along the path p is $S[p]x = U[p]x * \phi_J$. Further, let $p + \lambda = (\lambda_1, \lambda_2, \ldots \lambda_m, \lambda)$. This allows us to recursively define the next set of *propagated* and *scattering* coefficients by using $\tilde{W}$:

$$\tilde{W}U[p]x = \{S[p]x,\ U[p + \lambda]x\}_\lambda \tag{1.6.12}$$

which is shown in Figure 1.22

1.6.3 Resulting Properties

For ease, let us define the 'mth order scattering coefficients' as S_m which is the set of all coefficients with path length m. Further let S be the set of all scattering coefficients of any path length. The energy $\|Sx\|^2$ we then define as

$$\|Sx\|^2 = \sum_p \|S[p]x\|^2 \tag{1.6.13}$$

We can make W non-expansive with appropriate scaling. Further, define the energy $\|Wx\|^2$ as

$$\|Wx\|^2 = \|x * \phi\|^2 + \sum_\lambda \|x * \psi_\lambda\|^2 \tag{1.6.14}$$

then by Plancherel's formula

$$(1-\epsilon)\|x\|^2 \leq \|Wx\|^2 \leq \|x\|^2 \tag{1.6.15}$$

For the Morlet wavelets originally used in [65], $\epsilon = 0.25$, for the DT$\mathbb{C}$WT $\epsilon = 0$.

1.6.3.1 Translation Invariance

This is proven in section 2.4 of [88]. We have so far described the Scattering representation as being 'translation invariant for shifts up to 2^J'. We formalize this statement here.

For a 2-D averaging filter based on a father wavelet ϕ, $\phi_J = 2^{-J}\phi\left(2^{-J}\mathbf{u}\right)$ it is proven in Appendix B of [88] that shifting it by $\mathbf{c}$, which we denote as $\mathcal{L}_c$, is Lipschitz continuous:

$$\|\mathcal{L}_c\phi_J - \phi_J\| \leq 2^{-J+2}\|\nabla\phi\|_1|\mathbf{c}| \tag{1.6.16}$$

where $\|\nabla\phi\|_1$ is the ℓ_1 norm of the grad of ϕ.

For simplicity, let us define $A_J x = \phi_J * x$ and $Sx = A_J U x$. Then we get:

$$\begin{aligned}\|S\mathcal{L}_c x - Sx\| &= \|\mathcal{L}_c A_J U x - A_J U x\| &(1.6.17)\\ &\leq \|\mathcal{L}_c A_j - A_j\|\,\|Ux\| &(1.6.18)\\ &\leq 2^{-J+2}\|\nabla\phi\|_1|\mathbf{c}|\|x\| &(1.6.19)\end{aligned}$$

1.6.3.2 Stability to Noise

As W is non-expansive and the complex modulus is also non-expansive

$$\left\|\tilde{W}x - \tilde{W}y\right\| \leq \|x-y\| \tag{1.6.20}$$

As we have already shown that S is the repeated application of $\tilde{W}$ in (1.6.12), we can then say

$$\|Sx - Sy\| \leq \|x - y\| \tag{1.6.21}$$

making scattering non-expansive and stable to noise.

1.6.3.3 Stability to deformations

From [88], if $\mathcal{L}_\tau x = x(\mathbf{u} - \tau(\mathbf{u}))$ is a diffeomorphism which is bounded with $\|\nabla\tau\|_\infty \leq 1/2$, then there exists a $K_L > 0$ such that:

$$\|S\mathcal{L}_\tau x - Sx\| \leq K_L P F(\tau) \|x\| \tag{1.6.22}$$

where $P = \text{length}(p)$ is the scattering order, and $F(\tau)$ is a function of the size of the displacement, derivative and Hessian of τ, $H(\tau)$ [88]:

$$F(\tau) = 2^{-J} \|\tau\|_\infty + \|\nabla\tau\|_\infty \max\left(\log \frac{\|\Delta\tau\|_\infty}{\|\nabla\tau\|_\infty}, 1\right) + \|H(\tau)\|_\infty \tag{1.6.23}$$

1.6.3.4 Energy Decay

As $m \to \infty$ the invariant cofficients of path length m, U_m, decay towards zero:

$$\lim_{m\to\infty} U_m = 0 \tag{1.6.24}$$

This is an important property that suggests that we can stop scattering beyond a certain point. For image sizes on the order of a few hundred pixels by a few hundred pixels, $m = 3$ captures about 99% of the input energy. For many works using scattering transforms after [65] such as [68], [90], [91], setting $m = 2$ was found to be sufficient.

1.6.3.5 Number of Coefficients

While we have so far talked about non sampled signals $x(\mathbf{u})$, $\mathbf{u} \in \mathbb{R}^2$, in practice we want to apply scattering to sampled signals $x[\mathbf{n}]$, $\mathbf{n} \in \mathbb{Z}^2$. The averaging by ϕ_J means that we can subsample Sx by 2^J in each direction. However, now we need also need to index all the paths p that can be used to create the scattering coefficients. Limiting ourselves to $m = 2$ and using a wavelet transform with J scales and K discrete orientations the number of paths

for each S_m is the cardinality of the set p_m:

$$n(p_0) = 1 \tag{1.6.25}$$

$$n(p_1) = JK \tag{1.6.26}$$

$$n(p_2) = (J-1)K^2 + (J-2)K^2 + \ldots + K^2 \tag{1.6.27}$$

$$= \frac{1}{2}J(J-1)K^2 \tag{1.6.28}$$

The reason $n(p_2) \neq J^2K^2$ is due to the demodulating effect of the complex modulus. As $|x * \psi_\lambda|$ is more regular than $x * \psi_\lambda$, $|x * \psi_\lambda| * \psi_{\lambda'}$ is only non-negligible if $\psi_{\lambda'}$ is located at lower frequencies than ψ_λ. This means, we can discard over half of the scattering paths as their value will be near zero.

Summing up the above three equations and factoring in the reduced sample rate allowable due to averaging, for an input with N pixels, the scattering representation will have $\frac{N}{2^J}\left(1 + JK + \frac{1}{2}J(J-1)K^2\right)$ pixels.

References

[1] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[2] D. R. Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning", eng, *Neural Networks: The Official Journal of the International Neural Network Society*, vol. 16, no. 10, pp. 1429–1451, Dec. 2003.

[3] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't Decay the Learning Rate, Increase the Batch Size", *arXiv:1711.00489 [cs, stat]*, Nov. 2017. arXiv: 1711.00489 [cs, stat].

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[5] L. Bottou, "Stochastic Gradient Descent Tricks", en-US, vol. 7700, Jan. 2012.

[6] G. Montavon, G. Orr, and K.-R. Mller, *Neural Networks: Tricks of the Trade*, 2nd. Springer Publishing Company, Incorporated, 2012.

[7] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp", en, in *Neural Networks: Tricks of the Trade*, ser. Lecture Notes in Computer Science 7700, G. Montavon, G. B. Orr, and K.-R. Müller, Eds., Springer Berlin Heidelberg, 2012, pp. 9–48.

[8] Y. A. Ioannou, "Structural Priors in Deep Neural Networks", PhD thesis, Department of Engineering, University of Cambridge, Trumpington Street, Cambridge CB2 1PZ, United Kingdom, Sep. 2017.

[9] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", *arXiv:1412.6980 [cs]*, Dec. 2014. arXiv: 1412.6980 [cs].

[10] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[11] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method", *arXiv:1212.5701 [cs]*, Dec. 2012. arXiv: 1212.5701 [cs].

[12] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain [J]", *Psychol. Review*, vol. 65, pp. 386–408, Dec. 1958.

[13] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks", in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[14] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines", in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.

[15] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", in *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

[16] P. Lennie, "The cost of cortical computation", eng, *Current biology: CB*, vol. 13, no. 6, pp. 493–497, Mar. 2003.

[17] M. L. Minsky and S. A. Papert, *Perceptrons: Expanded Edition*. Cambridge, MA, USA: MIT Press, 1988.

[18] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators", *Neural Networks*, vol. 2, no. 5, pp. 359–366, Jan. 1989.

[19] G. Cybenko, "Approximation by superpositions of a sigmoidal function", en, *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, Dec. 1989.

[20] B. Widrow and M. E. Hoff, "Neurocomputing: Foundations of Research", in, J. A. Anderson and E. Rosenfeld, Eds., Cambridge, MA, USA: MIT Press, 1988, pp. 123–134.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1", in, D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, Eds., Cambridge, MA, USA: MIT Press, 1986, pp. 318–362.

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[23] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex", eng, *The Journal of Physiology*, vol. 160, pp. 106–154, Jan. 1962.

[24] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints", *International Journal of Computer Vision*, vol. 60, pp. 91–110, 2004.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", in *NIPS*, Curran Associates, Inc., 2012, pp. 1097–1105.

[26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", *arXiv:1409.0575 [cs]*, Sep. 2014. arXiv: 1409.0575 [cs].

[27] C. Cortes and V. Vapnik, "Support-vector networks", en, *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep. 1995.

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch", Oct. 2017.

[29] D. Mishkin, N. Sergievskiy, and J. Matas, "Systematic evaluation of CNN advances on the ImageNet", *arXiv:1606.02228 [cs]*, Jun. 2016. arXiv: 1606.02228 [cs].

[30] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", *arXiv:1409.1556 [cs]*, Sep. 2014. arXiv: 1409.1556 [cs].

[31] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely Connected Convolutional Networks", in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 2261–2269. arXiv: 1608.06993.

[32] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for Simplicity: The All Convolutional Net", *arXiv:1412.6806 [cs]*, Dec. 2014. arXiv: 1412.6806 [cs].

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 770–778. arXiv: 1512.03385.

[34] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks", *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5987–5995, 2017.

[35] S. Zagoruyko and N. Komodakis, "Wide Residual Networks", en, May 2016.

[36] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors", *arXiv:1207.0580 [cs]*, Jul. 2012. arXiv: 1207.0580 [cs].

[37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[38] Y. Gal and Z. Ghahramani, "Dropout As a Bayesian Approximation: Representing Model Uncertainty in Deep Learning", in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16, JMLR.org, 2016, pp. 1050–1059.

[39] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", *arXiv:1502.03167 [cs]*, Feb. 2015. arXiv: 1502.03167 [cs].

[40] Y. LeCun, C. Cortes, and C. Burges, "Modified NIST database of handwritted digits", http://yann.lecun.com/exdb/mnist/, 1998.

[41] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Tech. Rep., Apr. 2009.

[42] A. Krizhevsky, V. Nair, and G. Hinton, "CIFAR datasets", https://www.cs.toronto.edu/~kriz/cifar.html, 2009.

[43] F.-F. Li, "Tiny ImageNet Visual Recognition Challenge", Stanford cs231n: https://tiny-imagenet.herokuapp.com/, 2017.

[44] Stanford Vision Lab, "ImageNet CLS-LOC", https://www.kaggle.com/c/imagenet-object-localization-challenge/data, 2017.

[45] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes Challenge: A Retrospective", *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, Jan. 2015.

[46] Li Fei-Fei, R. Fergus, and P. Perona, "Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories", in *2004 Conference on Computer Vision and Pattern Recognition Workshop*, Jun. 2004, pp. 178–178.

[47] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper With Convolutions", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[48] P. L. Bartlett, S. N. Evans, and P. M. Long, "Representing smooth functions as compositions of near-identity functions with implications for deep network optimization", *arXiv:1804.05012 [cs, math, stat]*, Apr. 2018. arXiv: 1804.05012 `[cs, math, stat]`.

[49] P. L. Bartlett, D. P. Helmbold, and P. M. Long, "Gradient descent with identity initialization efficiently learns positive definite linear transformations by deep residual networks", *arXiv:1802.06093 [cs, math, stat]*, Feb. 2018. arXiv: 1802.06093 `[cs, math, stat]`.

[50] M. Holschneider and P. Tchamitchian, "Pointwise analysis of Riemann's "nondifferentiable" function", en, *Inventiones mathematicae*, vol. 105, no. 1, pp. 157–175, Dec. 1991.

[51] M. Vetterli and J. Kovacevic, *Wavelets and Subband Coding*, 2nd ed., ser. Prentice Hall Signal Processing Series. Prentice Hall PTR, 2007.

[52] J. Antoine, R. Murenzi, P. Vandergheynst, and S. Ali, *Two-Dimensional Wavelets and Their Relatives*. 2004.

[53] J. Kovacevic and A. Chebira, *An Introduction to Frames*. Hanover, MA, USA: Now Publishers Inc., 2008.

[54] I. W. Selesnick, R. G. Baraniuk, and N. G. Kingsbury, "The dual-tree complex wavelet transform", *Signal Processing Magazine, IEEE*, vol. 22, no. 6, pp. 123–151, 2005.

[55] S. Mallat, *A Wavelet Tour of Signal Processing*. Academic Press, 1998.

[56] R. R. Coifman and D. L. Donoho, "Translation-Invariant De-Noising", en, in *Wavelets and Statistics*, ser. Lecture Notes in Statistics 103, A. Antoniadis and G. Oppenheim, Eds., Springer New York, 1995, pp. 125–150.

[57] N. Kingsbury and J. Magarey, "Wavelet transforms in image processing", A. Prochazka, J. Uhlir, and P. Sovka, Eds., 1997.

[58] N. Kingsbury, "The Dual-Tree Complex Wavelet Transform: A New Technique For Shift Invariance And Directional Filters", in *1998 8th International Conference on Digital Signal Processing (DSP)*, Utah, Aug. 1998, pp. 319–322.

[59] ——, "The dual-tree complex wavelet transform: A new efficient tool for image restoration and enhancement", in *Signal Processing Conference (EUSIPCO 1998), 9th European*, Sep. 1998, pp. 1–4.

[60] N. Kingsbury, "Image processing with complex wavelets", *Philosophical Transactions of the Royal Society a-Mathematical Physical and Engineering Sciences*, vol. 357, no. 1760, pp. 2543–2560, Sep. 1999.

[61] ——, "Shift invariant properties of the dual-tree complex wavelet transform", in *Icassp '99: 1999 Ieee International Conference on Acoustics, Speech, and Signal Processing, Proceedings Vols I-Vi*, 1999, pp. 1221–1224.

[62] ——, "A dual-tree complex wavelet transform with improved orthogonality and symmetry properties", 2000.

[63] ——, "Complex wavelets for shift invariant analysis and filtering of signals", *Applied and Computational Harmonic Analysis*, vol. 10, no. 3, pp. 234–253, May 2001.

[64] J. Bruna and S. Mallat, "Classification with scattering operators", in *2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2011, pp. 1561–1566.

[65] ——, "Invariant Scattering Convolution Networks", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1872–1886, Aug. 2013.

[66] J. Bruna, "Scattering Representations for Recognition", Theses, Ecole Polytechnique X, Feb. 2013.

[67] E. Oyallon, S. Mallat, and L. Sifre, "Generic Deep Networks with Wavelet Scattering", *arXiv:1312.5940 [cs]*, Dec. 2013. arXiv: 1312.5940 [cs].

[68] E. Oyallon and S. Mallat, "Deep Roto-Translation Scattering for Object Classification", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2865–2873.

[69] L. Sifre and S. Mallat, "Rotation, Scaling and Deformation Invariant Scattering for Texture Discrimination", in *2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2013, pp. 1233–1240.

[70] L. Sifre and S. Mallat, "Rigid-Motion Scattering for Texture Classification", *arXiv:1403.1687 [cs]*, Mar. 2014. arXiv: 1403.1687 [cs].

[71] L. Sifre, "Rigid-Motion Scattering for Image Classification", PhD Thesis, Ecole Polytechnique, Oct. 2014.

[72] L. Sifre and J. Anden, *ScatNet*, École normale supérieure, Nov. 2013.

[73] P. de Rivaz and N. Kingsbury, "Bayesian image deconvolution and denoising using complex wavelets", in *2001 International Conference on Image Processing, 2001. Proceedings*, vol. 2, Oct. 2001, 273–276 vol.2.

[74] Y. Zhang and N. Kingsbury, "A Bayesian wavelet-based multidimensional deconvolution with sub-band emphasis", in *Engineering in Medicine and Biology Society*, 2008, pp. 3024–3027.

[75] G. Zhang and N. Kingsbury, "Variational Bayesian image restoration with group-sparse modeling of wavelet coefficients", *Digital Signal Processing*, Special Issue in Honour of William J. (Bill) Fitzgerald, vol. 47, pp. 157–168, Dec. 2015.

[76] M. Miller and N. Kingsbury, "Image denoising using derotated complex wavelet coefficients", eng, *IEEE transactions on image processing: a publication of the IEEE Signal Processing Society*, vol. 17, no. 9, pp. 1500–1511, Sep. 2008.

[77] S. Hatipoglu, S. K. Mitra, and N. Kingsbury, "Texture classification using dual-tree complex wavelet transform", in *Seventh International Conference on Image Processing and Its Applications*, 1999, pp. 344–347.

[78] P. de Rivaz and N. Kingsbury, "Complex wavelet features for fast texture image retrieval", in *1999 International Conference on Image Processing, 1999. ICIP 99. Proceedings*, vol. 1, 1999, 109–113 vol.1.

[79] P. Loo and N. G. Kingsbury, "Motion-estimation-based registration of geometrically distorted images for watermark recovery", P. W. Wong and E. J. Delp III, Eds., Aug. 2001, pp. 606–617.

[80] H. Chen and N. Kingsbury, "Efficient Registration of Nonrigid 3-D Bodies", *IEEE Transactions on Image Processing*, vol. 21, no. 1, pp. 262–272, Jan. 2012.

[81] J. Fauqueur, N. Kingsbury, and R. Anderson, "Multiscale keypoint detection using the dual-tree complex wavelet transform", in *Image Processing, 2006 IEEE International Conference On*, IEEE, 2006, pp. 1625–1628.

[82] R. Anderson, N. Kingsbury, and J. Fauqueur, "Determining Multiscale Image Feature Angles from Complex Wavelet Phases", en, in *Image Analysis and Recognition*, ser. Lecture Notes in Computer Science 3656, M. Kamel and A. Campilho, Eds., Springer Berlin Heidelberg, Sep. 2005, pp. 490–498.

[83] ——, "Rotation-invariant object recognition using edge profile clusters", in *Signal Processing Conference, 2006 14th European*, IEEE, 2006, pp. 1–5.

[84] P. Bendale, W. Triggs, and N. Kingsbury, "Multiscale keypoint analysis based on complex wavelets", in *BMVC 2010-British Machine Vision Conference*, BMVA Press, 2010, pp. 49–1.

[85] E. S. Ng and N. G. Kingsbury, "Robust pairwise matching of interest points with complex wavelets", *Image Processing, IEEE Transactions on*, vol. 21, no. 8, pp. 3429–3442, 2012.

[86] I. Selesnick, "Hilbert transform pairs of wavelet bases", *IEEE Signal Processing Letters*, vol. 8, no. 6, pp. 170–173, Jun. 2001.

[87] N. Kingsbury, "Design of Q-shift complex wavelets for image processing using frequency domain energy minimization", in *2003 International Conference on Image Processing, 2003. ICIP 2003. Proceedings*, vol. 1, Sep. 2003, I-1013-16 vol.1.

[88] S. Mallat, "Group Invariant Scattering", en, *Communications on Pure and Applied Mathematics*, vol. 65, no. 10, pp. 1331–1398, Oct. 2012.

[89] I. Waldspurger, A. d'Aspremont, and S. Mallat, "Phase Recovery, MaxCut and Complex Semidefinite Programming", *arXiv:1206.0102 [math]*, Jun. 2012. arXiv: 1206.0102 [math].

[90] E. Oyallon, "A Hybrid Network: Scattering and Convnet", 2017.

[91] E. Oyallon, E. Belilovsky, and S. Zagoruyko, "Scaling the Scattering Transform: Deep Hybrid Networks", in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 5619–5628. arXiv: 1703.08961.