UNIVERSITY OF CAPE TOWN

Deep Neural Networks for Video Classification in Ecology

Author:
Alexander CONWAY

Supervisor:
Dr. Ian DURBACH

A half-dissertation submitted in fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in the

Department of Statistics

Declaration of Authorship

I, Alexander CONWAY, declare that this thesis titled, "Deep Neural Networks for Video Classification in Ecology" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

_____

Date:

_____

Abstract

Analyzing large volumes of video data is a challenging and time-consuming task. Automating this process would very valuable, especially in ecological research where massive amounts of video can be used to unlock new avenues of ecological research into the behaviour of animals in their environments.

Deep Neural Networks, particularly Deep Convolutional Neural Networks, are a powerful class of models for computer vision. When combined with Recurrent Neural Networks, Deep Convolutional models can be applied to video for frame level video classification.

This research studies two datasets: penguins and seals. The purpose of the research is to compare the performance of image-only CNNs, which treat each frame of a video independently, against a combined CNN-RNN approach; and to assess whether incorporating the motion information in the temporal aspect of video improves the accuracy of classifications in these two datasets.

Video and image-only models offer similar out-of-sample performance on the simpler seals dataset but the video model led to moderate performance improvements on the more complex penguin action recognition dataset.

Acknowledgements

I would like to express my heartfelt gratitude to Dr. Ian Durbach for his patience, kindness and always valuable suggestions.

I would also like to thank my mother, grandparents and friends for their support.

Finally, I would like to thank Google Cloud Platform for providing the computing resources used in this research.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

3DCNN	**3**-**D**imensional **C**onvolutional **N**eural **N**etwork
ANN	**A**rtificial **N**eural **N**etwork
AVEDs	**A**nimal-borne **V**ideo and **E**nvironmental **D**ata collection systems
BPTT	**B**ack **P**ropagation **T**hrough **T**ime
CNN	**C**onvolutional **N**eural **N**etwork
DNN	**D**eep **N**eural **N**etwork
FPS	**F**rames **P**er **S**econd
GPU	**G**raphics **P**rocessing **U**nit
GRU	**G**ated **R**ecurrent **U**nit
ILSVRC	**I**magenet **L**arge **S**cale **V**isual **R**ecognition **C**hallenge
LRCN	**L**ong-term **R**ecurrent **C**onvolutional **N**etwork
LSTM	**L**ong **S**hort **T**erm **M**emory
MLP	**M**aximum **L**ikelihood **E**stimation
MLP	**M**ulti **L**ayer **P**erceptron
NN	**N**eural **N**etwork
ReLU	**R**ectified **L**inear **U**nit
RGB	**R**ed **G**reen **B**lue
RMSE	**R**oot **M**ean **S**quare **E**rror
RNN	**R**ecurrent **N**eural **N**etwork
SIFT	**S**cale **I**nvariant **F**eature **T**ransform
SGD	**S**tochastic **G**radient **D**escent
SVM	**S**upport **V**ector **M**achine

Chapter 1

Introduction

1.1 Background

Ecology is the branch of biology that studies the relations of organisms to one another and to their environments. Photography has long been used to document organisms in their natural habitats and gain insights into the behaviour of animals that cannot be gained in any other way.

A camera trap is a camera triggered by a trip wire, motion or infrared sensor. Camera traps are a non-intrusive way to study animals that would be scared off or caused to behave differently under human observation (Swinnen et al., 2014), (Bowley et al., 2017). They have been used extensively in ecological research, for example to evaluate spatio-temporal inter-species dynamics and habitat occupancy in the Serengeti National Park in Tanzania (Anderson et al., 2016) and as a non-destructive tool to assess fish populations in clearwater Amazonian rivers where netting techniques have been found to under-represent the true populations (Reis-filho and Jose, 2017).

FIGURE 1.1: The earliest nighttime flash photograph using a trip wire, taken by George Shiras in the 1890s (Wender, 2015)

Unlike camera traps which remain fixed in place and only capture footage of animals that happen to enter their field of view, animal-borne cameras are attached to animals to collect point-of-view imagery that allows the viewer to see the world through the eyes of the animal. Crittercam is the name of the first animal-borne integrated data logging system that was invented by National Geographic to record video, audio and other sensor data for studying the behaviour of large marine vertebrates at sea (Marshall, 1998).

(A) A feral cat wearing a collar with a modified GoPro Hero 3 video camera and tracking device (Mcgregor et al., 2015)

(B) A tiger shark with a fin-attached Crittercam (Heithaus et al., 2001)

(C) A Crittercam attached using a harness used to investigate the sub-ice foraging behaviour of emperor penguins (Ponganis et al., 2000)

FIGURE 1.2: Examples of animal-borne video cameras used to record animal behaviour

Animal-borne cameras have revealed facts about animal behaviour that otherwise could not have been known (Moll et al., 2007). Crittercam and other animal-borne video recording systems have been used extensively to study foraging ecology (e.g. feral cats hunt frogs significantly more than previously thought from stomach analysis (Mcgregor et al., 2015)) and habitat use (e.g. tiger sharks use shallow habitats twice as much as previously thought (Heithaus et al., 2001)). Commercial lobster and deepwater fishing licenses were granted near the largest remaining colony of endangered Hawaiian monk seals based on scat analysis studies that suggested the seals foraged in shallow waters only. A bias caused by differential digestion of animal matter underestimated the importance of deepwater foraging in the seals' diets and a study using animal-borne video cameras with depth sensors led to a United States federal court closing the fisheries and charging them with a violation of the Endangered Species Act for failing to assess the impacts on the endangered seals. Weddel seals were similarly fitted with video recording devices and observed to expel blasts of air from their nostrils to flush small fish burrowed in sub-ice crevices, a previously unknown hunting behaviour (Davis et al., 1999).

Technological advances in better camera quality, smaller camera size, longer battery life and cheaper storage accelerated by the development of consumer smart-phones have enabled animal-borne cameras to record more data at better quality on a broader variety of

animals, becoming small enough to deploy on birds in flight (Rutz, Weir, and Kacelnik, 2007). Although pictures are more commonly used in camera trap studies and are easier to work with (Swinnen et al., 2014), video can provide richer insight into animal behaviour and the motion information contained in video can significantly improve animal identification in camera trap footage (Trinh et al., 2016).

Footage captured using camera traps or animal-borne video cameras needs to be annotated for use in scientific research. The current process of annotating footage is very labour intensive requiring highly trained scientists to manually annotate the content of images and videos (frame by frame), presenting a major bottleneck for scientific research based on this data (Kelly, 2001; Walther, Edgington, and Koch, 2004) and necessitating the development of computer-assisted approaches (Lahiri et al., 2011). Gray et al. (2019) use CNNs to identify whales from drone footage and Ridge et al. (2020) use CNNs to identify Oyster reefs from aerial imagery. Several attempts have been made to automatically identify animals in camera trap footage, for example using Scale Invariant Feature Transform (SIFT) (Lowe, 1999) features with a Support Vector Machine (SVM) classifier (Chen et al., 2014). Some research has also been conducted on automatically analyzing the behaviour of animals, for example using AdaBoost classifiers with Haar features to identify lion heads with head locations used to classify the lion behaviour as keeping still, walking or trotting (Burghardt and Janko, 2006).

These approaches require careful feature engineering that is problem specific and is not transferable to detecting different animals or behaviours in different settings and they have had limited success.

1.2 Deep Neural Networks

A Deep Neural Network (DNN) is a biologically inspired machine learning model that uses stacked non-linear combinations of inputs together with a gradient descent learning procedure to jointly learn feature representations together with a classifier based on labeled data. DNNs are the current state-of-the-art for image classification (Hu, Shen, and Sun, 2018), video classification (Jiang et al., 2018), text classification (Howard and Ruder, 2018), speech recognition (Graves, Mohamed, and Hinton, 2013), and many other challenging perceptual problems with very high dimensional inputs where hand-designing input feature representations is nontrivial (Liu et al., 2016).

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual image classification challenge with an associated classification dataset containing millions of images spanning a thousand disjoint classes. In 2011, the best result was a top-5 error rate of 26% from a model using an ensemble of hand-designed and generic feature extraction

methods using Fisher vectors (Sanchez, Perronnin, and Akata, 2011). In 2012, a type of DNN called a Convolutional Neural Network (CNN) model won ILSVRC for the first time with a significant reduction of top-5 error rate to 17% (Krizhevsky, Sutskever, and Hinton, 2012). Since then, every ImageNet-winning model has been a CNN. Average human performance was found to be a top-5 error rate of 5.1% (Russakovsky, Deng, Su, Krause, Satheesh, Ma, Huang, Karpathy, Khosla, Bernstein, et al., 2015b). The winning entry in 2017 had a top-5 error rate of 2.25% (Hu, Shen, and Sun, 2018).

The success of DNNs is due to a combination of increases in the amount of data available, significant increases in the amount of computational resources available (the 2012 ILSVRC winner was implemented on high-powered Graphical Processing Unit (GPU) devices), and a plethora of machine learning insights into regularization, sparsity and optimization (Socher, 2014). Researchers releasing their software implementations and trained neural network architectures has also accelerated the development of deep learning, in part because the feature extraction layers of a CNN can be used on entirely new datasets (usually with some additional training) in a process called "transfer learning" (Goodfellow, Bengio, and Courville, 2016).

There are two approaches to using DNNs for video classification beyond treating the problem as an image classification task by modeling frames independently. The first approach models the spatial information in frames separately from the temporal information, for example a Long-term Recurrent Convolutional Network (LRCN) uses a ILSVRC-trained CNN image classification model as a feature extractor on video frames and then fits a type of Recurrent Neural Network (RNN) called a Long-Short Term Memory (LSTM) network on top of the sequence of extracted feature vectors to model the time component of video (Donahue et al., 2014). This approach has been used successfully in ecology by Trinh et al. (2016) to detect birds flying into wind turbines that are not clearly identifiable in isolated video frames. The other approach to video classification does not leverage fine-tuning from ImageNet but instead jointly models the spatial and temporal components of video using 3-D convolutional models that operate on and between frames at the same time (**Ji2013**), (Tran et al., 2015).

DNNs have been applied in a handful of ecological studies. Chen et al. (2014) uses a custom DNN with no fine-tuning to classify animals in camera trap footage into one of 20 species with 38% accuracy compared to a 33% baseline. Gomez Villa, Salazar, and Vargas (2017) fine-tune 3 ILSVRC-winning CNNs to classify animals in camera trap footage into one of 26 classes with 89% accuracy (Zhang et al., 2016). Weinstein (2018) fine-tune a CNN to detect objects in camera trap footage with some success. Siddiqui et al. (2018) classify underwater camera trap video clips by classifying the frames in isolation using fine-tuned

CNNs. Zhang et al. (2016) develop a cross-frame patch verification method on top of DNN extracted frame features after finding that using a single image is not sufficient for accurate animal background classification.

1.3 Research Objectives

This study will explore the application of deep neural networks to video classification in ecology on two datasets: a camera trap video dataset and an animal-borne video dataset.

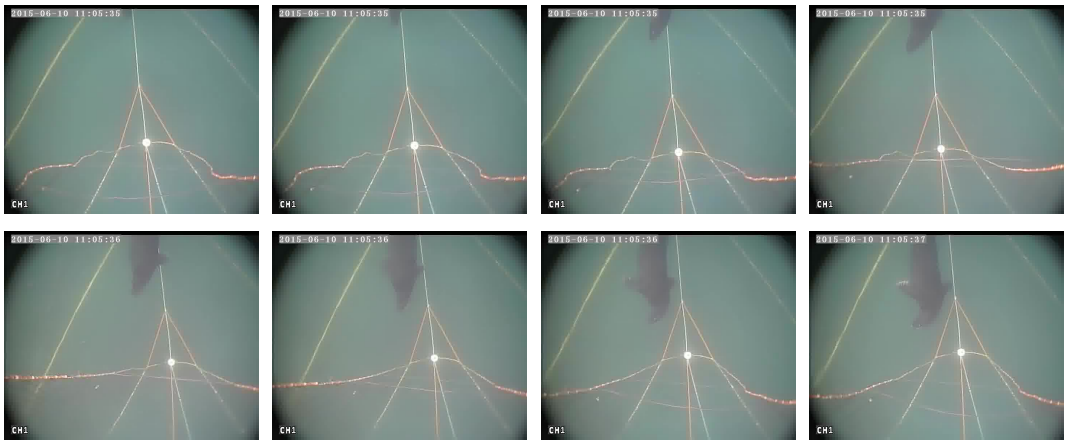


FIGURE 1.3: A clip of 8 continuous frames from one of the videos in the seals underwater camera trap video dataset used in this study

The camera trap dataset is an underwater camera mounted on a fishing vessel that comes from a trial investigating the effects of using an acoustic deterrent device to deter seals in order to prevent them from becoming stuck in the trap nets being dragged behind the boat. The seals dataset contains 3.5 hours of annotated video.

The other dataset is an animal-borne video dataset with video cameras mounted to Penguins swimming in the ocean with the goal is to classify the penguin behaviour in each frame into one of seven classes: search (above the water surface), subsurface, shallow, descent, bottom, ascent, breath. The penguins dataset contains 9 hours of annotated video.

FIGURE 1.4: A clip of 8 continuous frames from one of the videos in the penguins animal-borne video dataset used in this study

To the best of our knowledge, this is the first time DNNs have been applied to classifying animal-borne video. If video classification can be accomplished accurately using DNNs then this could save researchers significant amounts of time by allowing them to label a relatively small subset of data and then use a DNN to predict labels for much larger amounts of data. This could potentially open new avenues for research and allow researchers to answer previously unanswerable research questions.
This research aims to determine the extent to which DNNs can be used for video classification in ecology and explore which kinds of DNN models work best for classifying camera trap and animal-borne video data in ecology.

1.4 Outline of dissertation

This dissertation is laid out in seven chapters, including this introductory chapter. Chapter 2 establishes the theory of artificial neural networks, beginning with multi-layer perceptrons and progressing to convolutional and recurrent neural networks. Chapter 3 begins with a discussion of the ImageNet Large Scale Visual Recognition Challenge and then presents several deep neural network architectures that have been successfully applied to image classification problems. Chapter 4 presents several deep neural network architectures for video classification. Chapter 5 gives a description of the two video datasets used in this study and describes the implementation details of the software package developed to classify the frame streams of the two datasets. Chapter 6 presents the results and finally Chapter 7 concludes and suggests directions for future work.

Chapter 2

Basic Neural Network Theory

This chapter introduces neural networks by defining feedforward fully-connected neural networks together with common activation functions, cost functions, backpropagation, details on how to train a network and regularization.

2.1 Feedforward "Fully-connected" Neural Networks

A feedforward neural network, also known as a multilayer perceptron (MLP), is mathematical model that takes the form of a directed acyclic graph with at least 3 layers of stacked nodes (called "neurons") connected by edges (called "weights") arranged in a network (Rosenblatt, 1962).

These models are called feedforward networks because information flows through the network from its first layer (called the *input* layer) through intermediate computations and finally to its final layer (called the *output* layer) with no feedback or recurrent connections. Neural networks with feedback connections are called recurrent neural networks and will be discussed in Chapter 4. Feedforward neural networks, parameterized by their weights $\mathbf{W}$, are used to approximate some function $f(\mathbf{x}; \mathbf{W})$ given some inputs $\mathbf{x}$. For example, for a k-class classifier $p(\mathbf{y}|\mathbf{x}, \mathbf{W})$ maps an input vector $\mathbf{x}$ onto a probability distribution $\hat{\mathbf{y}}$ where the $\hat{y}_k$ is the networks' predicted probability of the input belonging to class k.

A feedforward neural network is comprised of artificial neurons arranged in "fully-connected" or "dense" layers with each neuron in one layer connected to every neuron in the subsequent layer. The output value of a neuron in one layer is the input in the following layer until the final output layer. Hidden layers are the layers between the input (first) and output (final) layers. A deep neural network (DNN) is a neural network with at least 2 hidden layers.

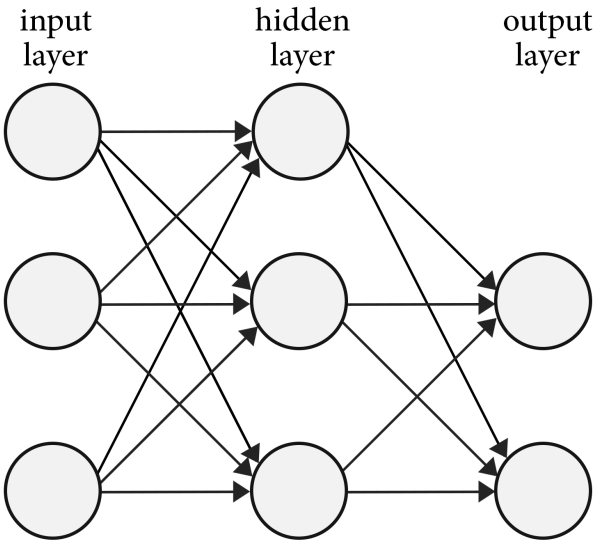


FIGURE 2.1: An example two layer feedforward neural network architecture with three inputs, a single hidden layer with three neurons, and two output neurons. When describing the number of layers in a network, the number of layers with trainable weights are counted.

Mathematically, a feedforward neural network is defined as follows:

$$z_j^{(l)} = g\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} a_i^{(l-1)}\right) \tag{2.1}$$

where:

$l =$ layer index with $l = 0, \ldots, L$.

$l = 0$ is designated the input layer with value x

$l = L$ is designated the output layer with value y

$d^l =$ the dimensionality of (number of neurons in) layer l

$x_i =$ the i^{th} input value where $\mathbf{x} = x_1, x_2, ..., x_n$

$b_j^l =$ the bias of the j^{th} neuron in the l^{th} layer

$w_{ij}^l =$ weight from the i^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer

$z_j^l =$ the weighted input or "pre-activation" of the j^{th} neuron in the l^{th} layer

$a_j^l =$ the output "activation" of the j^{th} neuron in the l^{th} layer

$g(\cdot) =$ non-linear activation function applied element-wise

$y = f(x, W)$ is the function defined by the neural network

2.1.1 Artificial Neurons

Artificial neurons are biologically inspired computational units (Goodfellow, Bengio, and Courville, 2016) that, like a biological neurons, process and transmit information. A neuron processes information by computing computing the linear combination of its inputs and

then applying a non-linear activation function to compute the neuron's outputs. In a neural network, a neuron's output becomes the input into any neurons connected to it in the subsequent layer of the network.

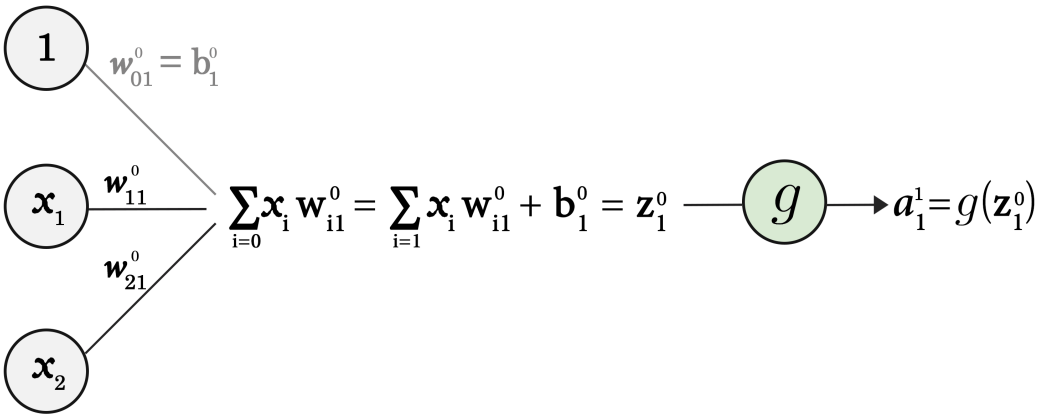


FIGURE 2.2: An artificial neuron with labeled inputs $\mathbf{x}_i$, weights $\mathbf{w}_{ij}$, bias $\mathbf{b}_j$, pre-activation $\mathbf{z}_j$, activation $\mathbf{a}_j$ and activation function g

The first artificial neuron model, developed by Frank Rosenblatt in the 1950s, was called the perceptron (Rosenblatt, 1962). The perceptron takes several binary inputs $x_1, x_2, \ldots, x_n$ and produces a single binary output. It is parameterized by real-valued weights $w_1, w_2, \ldots, w_n$ that express the importance of the respective inputs to the output. The output is computed by applying a threshold function to the weighted sum of the inputs and weights:

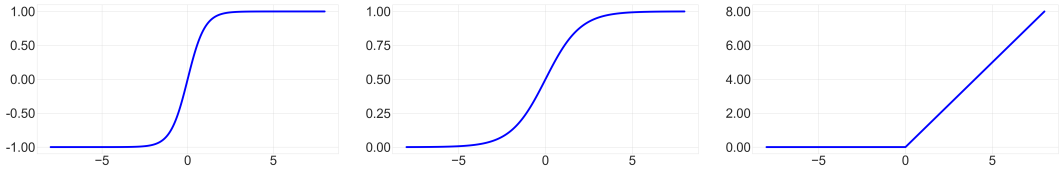
$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases} \tag{2.2}$$

This can be re-written by re-arranging and replacing the threshold with a "bias" term $b = -$threshold which is represented in the artificial neuron model as a constant input 1 and measures how difficult it is for the perceptron to return a positive value ("fire" in the language of biological neurons) (Goodfellow, Bengio, and Courville, 2016).

Small changes in the inputs or weights of any single perceptron in a network of perceptrons can cause the output to completely change as the threshold is passed. Sigmoid neurons are similar to perceptrons but relax the constraint that the inputs have to be binary and instead accept inputs with any value between 0 and 1. Sigmoid neurons also replace the threshold activation function of the perceptron with the sigmoid activation function (described in the next section) so that small changes in the input of the sigmoid neuron lead to small changes in its output. Modern neural networks accept any real-numbered input and use a range of different activation functions.

2.1.2 Activation Functions

Activation functions are always non-linear because a multi-layer neural network with linear activation functions could be re-parameterized as a single layer network due to the fact that the product of weight matrices $W_1, W_2, \ldots W_N$ could be rewritten as W^*. Non-linear activation functions enable neural networks to learn complex mappings from input to outputs. They also typically have derivatives that are easy to compute to speed up the process of learning optimal network parameters during backpropagation.



(A) Tanh (B) Sigmoid (C) ReLU

FIGURE 2.3: Common Activation Functions

Sigmoid

The sigmoid activation function (also called the logistic function) maps any real-valued input x into the range $[0, 1]$:

$$g(x) = \frac{1}{(1 + e^{-x})} \iff g'(x) = g(x)(1 - g(x)) \tag{2.3}$$

The sigmoid activation function was very popular in the early neural network literature because they tend to perform well when networks are very small but they are seldom used today. (Goodfellow, Bengio, and Courville, 2016). They exhibit 3 issues. First, sigmoid outputs are centered at 0.5 so if all the components of an input vector ar positive then all of the weight updates will have the same sign and will all increase or decrease together which slows down network convergence (Bishop, 2006). Second, sigmoid activations "saturate" in the sense that large and very large positive or negative pre-activation values produce activation values near 0 or 1. This is a problem because, as we will see in the section on backpropagation, this implies that the weights in these neurons do not update and neurons connected to saturated neurons only update slowly. Finally, in contrast to the ReLU activation function, the exponential function is computationally expensive to compute in comparison to a piecewise linear function. (Nielsen, 2015)

Hyperbolic tangent

The hyperbolic tangent (tanh) activation function maps any real-valued input x into the range $[-1, 1]$:

$$g(x) = \frac{1}{(1 + e^{-x})} \iff g'(x) = 1 - g(x)^2 \tag{2.4}$$

The tanh function is sigmoidal and thus experiences the same saturating and computationally expensive issues as the sigmoid function but it is centered at zero and as such is always preferable to the sigmoid activation function (Goldberg, 2017).

Rectified Linear Unit

The ReLU unit clips any input value $x < 0$ at 0:

$$g(x) = max(0, x) \iff g'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases} \tag{2.5}$$

The Rectified Linear Unit (ReLU) activation function is a simple piecewise lienar activation function that does not saturate like the sigmoid and tanh and is much faster to compute. It is the most commonly used activation function in modern ANN architectures. It was introduced in early neural network models and dates at least as far as the Neocognitron (Fukushima, 1980) but it was largely ignored in favour of the sigmoid due to a belief that activation functions with non-differentiable points must be avoided (Goodfellow, Bengio, and Courville, 2016). It was popularized by Nair and Hinton (2010) and Krizhevsky, Sutskever, and Hinton (2012) whose AlexNet CNN architecture (Krizhevsky, Sutskever, and Hinton, 2012) was a breakthrough improvement over the state-of-the-art in image classification in 2012, almost halving the previous error rate in the ImageNet classification challenge. The use of ReLU activation functions was deemed by the AlexNet authors to be the most important factor in the success of their model.

2.1.3 Output Units

The neurons in the output layer of a neural network typically have a different activation function to the neurons in the previous layers. In the case of regression where the network seeks to predict a real-valued continuous numerical output, the output layer typically contains a single neuron with a linear activation function. In a 1-of-K class classification context, the desired output is a probability distribution over the K classes being predicted and the softmax function (which can be seen as a multi-class version of the sigmoid function) is

used to transform a network's real-valued activation values into a probability distribution with values between 0 and 1 that together sum to 1. The softmax function $\mathbb{R}^K \longmapsto \mathbb{R}^K$ applied to a vector $\mathbf{a} = a_1, \ldots, a_K$ is defined as:

$$g(a_i) = \frac{e^{a_i}}{\sum_{k=1}^{K} e^{a_k}} \tag{2.6}$$

2.2 Forward propagation

The process by which information propagates and flows forward through a feedforward neural network from the inputs $\mathbf{x}$ to produce a predicted output $\hat{\mathbf{y}}$ is called *forward propagation*.

Algorithm 1 The Forward propagation Algorithm

1: **Input:** Vector of inputs $\mathbf{x}$
2: **Input:** Neural network model function $f(\mathbf{x}; W)$ parameterized by $\mathbf{W}$
3: **Input:** Activation function $g(\cdot)$
4: **for** l in $1, 2, \ldots, L$ **do**
5: Compute neuron pre-activations $z_j^{(l)} = \sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} a_i^{(l-1)}$
6: Compute neuron activations a $a_j^{(l)} = g(z_j^{(l)})$
7: **return** $\hat{\mathbf{y}} = \mathbf{a}^{(\mathbf{L})}$

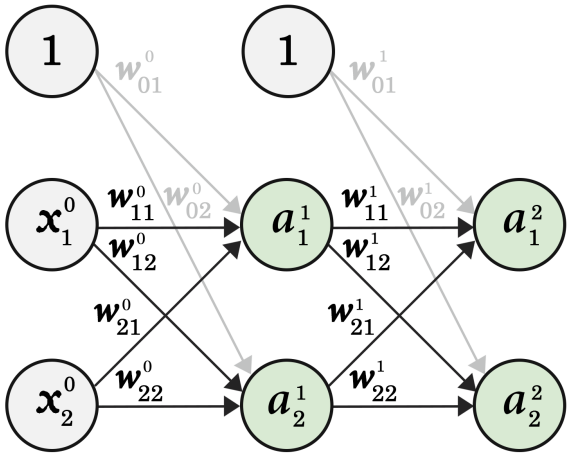


FIGURE 2.4: An example forward propagation algorithm calculation

2.3 Cost Functions

A cost function, also known as an "objective", "loss" or "error" function, measures how well a predictive model's outputs correspond to known ground truth observed values. We will

see in the next two sections how a neural network's weights can be updated using back-propagation and an optimization algorithm to make the network produce more accurate predictions and central to this process is the concept of a cost function.

Cost functions can take many forms, for example the mean-squared between observations and prediction values is a commonly used cost function for regression problems. A cost function $C_x(\hat{\mathbf{y}}, \mathbf{y})$ maps pairs of model output predictions $\hat{\mathbf{y}}_\mathbf{i}$ and ground-truth known values $\mathbf{y}_\mathbf{i}$ to a single scalar number that should be greater than or equal to zero and return low values when a model's predictions are accurate and higher values otherwise. The cost is computed for each observation in a dataset $\mathbf{x}_i$ and then averaged to calculate an overall cost:

$$C = \frac{1}{n} \sum_i C_{x_i} \tag{2.7}$$

2.3.1 Cross-entropy Cost

The choice of cost function is closely related to the choice of output units and for classification problems with a softmax function in the final layer, the categorical cross-entropy cost function is typically used (Nielsen, 2015).

The categorical cross-entropy cost function for a K-class classification problem with N observations is defined as follows:

$$C = -\frac{1}{n} \sum_{n=1}^{N} \sum_{k=1}^{K} [y_k \ln \hat{y}_k + (1 - \hat{y}_k) \ln (1 - \hat{y}_k)] \tag{2.8}$$

The categorical cross-entropy cost function derives from the principle of maximum likelihood estimation (MLE) in Statistics which is a method of using data observations to estimate the parameters of a model. The likelihood function $\mathcal{L}(\theta|x)$ for a discrete random variable is a function of the parameters of a model θ given the data $\mathbf{x}$ defined as follows:

$$\mathcal{L}(\theta|x) = p_\theta(x) = P_\theta(X = x) = \prod_{n=1}^{N} p\left(\mathbf{x}_i; \boldsymbol{\theta}\right) = \prod_{n=1}^{N} \prod_{k=1}^{K} p(y_n = k|\mathbf{x_n}, \mathbf{W}) \tag{2.9}$$

The method of maximum likelihood finds the model parameter values θ called a maximum likelihood estimate which is the set of parameter values that maximise the likelihood function in the sense that they make the data most probable given the model. The MLE estimate $\hat{\theta}$ out of the space of all possible parameter settings Θ is defined as follows:

$$\hat{\theta} \in \{\arg\max_{\theta \in \boldsymbol{\Theta}} \mathcal{L}(\theta; x)\} \tag{2.10}$$

For some models, a MLE estimate can be computed using a closed-form solution derived by analytically maximising the likelihood function given a parametric model (for example a binary outcome modeled using a binomial distribution). However, for many models, no closed-form analytic solution exists and an MLE can only be found via numerical optimization. In practice, it is often convenient to maximise the natural logarithm of the likelihood function, called the log-likelihood $\ell(\theta; x)$, rather than maximising the likelihood function directly which is equivalent because the log function is a strictly increasing function.

$$\ell(\theta; \mathbf{x}) = \ln \mathcal{L}(\theta; \mathbf{x}) \tag{2.11}$$

If the data are independent and identically distributed then the sample log-likelihood function for the can be written as follows:

$$\hat{\ell}(\theta; \mathbf{x}) = \frac{1}{n} \sum_{n=1}^{N} \sum_{k=1}^{K} \ln p\left(y_n = k | \mathbf{x}_i, \theta\right) \tag{2.12}$$

2.4 Backpropagation

The process of training a neural network involves initializing its weights randomly, using forward propagation to compute the network's outputs given the data, measuring the errors in these outputs using a cost function, and then iteratively updating the weights to reduce the error of the network (Rumelhart, Hinton, and Williams, 1986).

Backpropagation, which is shorthand for "the backward propagation of errors", is the name for the method of recursively applying the chain rule for differentiation to iteratively compute gradients of the cost ("error") function with respect to each of the weights in the network, starting from the output layer and moving backward through the hidden layers of the network(Friedman, Hastie, and Tibshirani, 2001). Backpropagation allows the information from the cost computed given the outputs in the final layer of the network to flow back through the network by providing a way to calculate the gradients of the cost with respect the weights not only in the output layer but also in the hidden layers which are not directly connected to the output (Goodfellow, Bengio, and Courville, 2016).

In the general case with vectors $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^m$ and functions $g : \mathbb{R}^m \mapsto \mathbb{R}^n$ and $f : \mathbb{R}^n \mapsto \mathbb{R}$, if $\mathbf{y} = g(\mathbf{x})$ and $z = f(\mathbf{y}) = f(g(\mathbf{x}))$ then the chain rule of differentiation is defined as follows:

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \tag{2.13}$$

Defining the intermediate quantity δ_j^l to represent the *error* in the j^{th} neuron in the l^{th} layer, backpropagation uses the chain rule of differentiation to compute the error δ_j^l for each neuron which is then used to compute the gradient of the network's cost function C with respect to the network's weights $\{w_{ij}^l\}$ given an arbitrary activation function g as follows:

$$\frac{\partial C}{\partial w_{ij}^{(l)}} = \frac{\partial C}{\partial a_j^{(l)}} \times \frac{\partial a_j^{(l)}}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} \times x_i^{(l-1)} \text{ for all } i, j, l \tag{2.14}$$

The error in the output layer is defined as:

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \times \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \times g'\left(z_j^L\right) \tag{2.15}$$

And the error in the layers is defined recursively as:

$$\delta_i^{l-1} = \frac{\partial C}{\partial z_i^{l-1}} = \frac{\partial a_i^{l-1}}{\partial z_i^{l-1}} \times \frac{\partial C}{\partial a_i^{l-1}} \tag{2.16}$$

$$= g'\left(z_i^{l-1}\right) \sum_{j=0}^{d^l} \frac{\partial C}{\partial z_j^l} \times \frac{\partial z_j^l}{\partial a_j^l} \tag{2.17}$$

$$= g'\left(z_i^{l-1}\right) \sum_{j=0}^{d^l} \delta_j^l \times w_{ij}^l \tag{2.18}$$

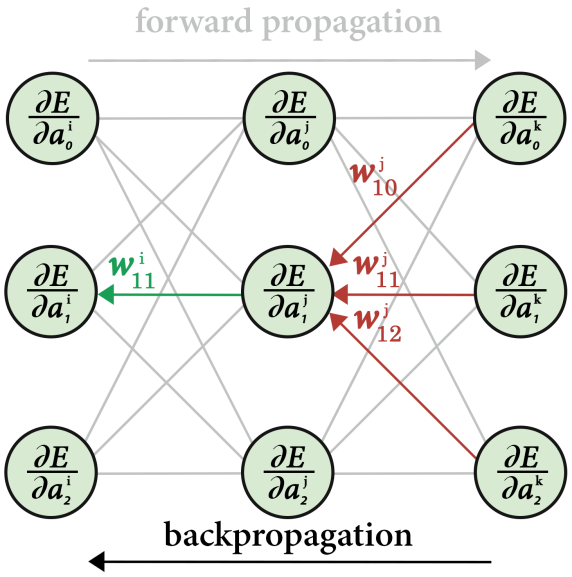


FIGURE 2.5: An example backpropagation algorithm calculation

Using the $\odot$ Hadamard product symbol to indicate element-wise multiplication and applying activation function g element-wise, the back-propagation algorithm can be summarized as follows:

Algorithm 2 The Backpropagation algorithm

1: **Input:** Vector of inputs $\mathbf{x}$ with a corresponding ground-truth label y
2: **Input:** Neural network model function $f(\mathbf{x}; W)$ parameterized by $\mathbf{W}$
3: **Input:** Activation function $g(\cdot)$
4: **Input:** Cost function C
5: Compute current prediction using forward propagation $\mathbf{a}^L = \hat{\mathbf{y}}_\mathbf{i} = f(\mathbf{x}_i; \mathbf{W})$
6: Compute cost $C(\hat{\mathbf{y}}_\mathbf{i}, \mathbf{y}_\mathbf{i})$
7: Compute error for each neuron in output layer $\boldsymbol{\delta}^L = \nabla_{\mathbf{a}^L} C \odot g'\left(\mathbf{z}^L\right)$
8: Compute weight gradients in output layer $\nabla_{\mathbf{W}^L} C = \delta^{(k)} \mathbf{a}^{(k-1)}$
9: **for** l in $L-1, L-2, \ldots, 1$ **do**
10: Compute error for each neuron in l^{th} layer $\boldsymbol{\delta}^l = \mathbf{W}^{l+1}\boldsymbol{\delta}^{l+1} \odot g'\left(\mathbf{z}^l\right)$
11: Compute weight gradients for neurons in l^{th} layer $\nabla_{\mathbf{W}^l} C = \boldsymbol{\delta}^l \mathbf{a}^{l-1}$
12: **return** Weight gradients $\nabla_\mathbf{W} C$

2.5 Gradient Descent & Optimization

Gradient descent is an iterative numerical optimization algorithm for finding the local minimum of a function by taking steps in the function's parameter space proportional to the negative of the function's gradient at the current point.

Training a neural network involves initializing its weights randomly and then iteratively updating the weights using gradient descent on the network's cost function using gradients computed using backpropagation. As the neural network's weights are updated, its hidden units which are not part of the input or output come to encode important feature transformations of the input that enable the network to more accurately produce outputs (Rumelhart, Hinton, and Williams, 1986). In gradient descent, the gradient of the cost function computed over the entire dataset is used so in practice a variant called Stochastic Gradient Descent (SGD) which samples subsets of the dataset to compute gradients and tends to converge faster than gradient descent is typically used.

The goal of the SGD algorithm is to update the network's parameters W to minimize the total cost $\sum_{i=1}^{n} C(\hat{\mathbf{y}}, \mathbf{y}_\mathbf{i})$ over the set of observation. It works by repeatedly sampling a random training example and computing the gradient of the cost of the example with respect to the parameters W (line 9). The network's parameters are then updated in the opposite direction of the gradient by multiplying the gradient by a small factor controlled by a parameter called the η that controls the size of the steps taken during gradient descent. A training epoch is defined as a set of SGD steps during which each training observation has been used exactly once to compute gradients.

Algorithm 3 Stochastic Gradient Descent for Training a Neural Network

Input: Training dataset of n input/output pairs $\{(\mathbf{x_1}, \mathbf{y_1}), \ldots, (\mathbf{x_n}, \mathbf{y_n})\}$
Input: Neural network model function $f(\mathbf{x}; W)$ parameterized by $\mathbf{W}$
Input: Cost function C
Input: Learning rate η
for $epoch$ in $1, 2, \ldots, epochs$ **do**
    Sample a single training dataset observation $(\mathbf{x_i}, \mathbf{y_i})$
    Compute current prediction using forward propagation $\hat{\mathbf{y}}_\mathbf{i} = f(\mathbf{x_i}; \mathbf{W})$
    Compute the cost $C(\hat{\mathbf{y}}_\mathbf{i}, \mathbf{y_i})$
    Compute gradients using backpropagation $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} \leftarrow$ gradients of $C(\hat{\mathbf{y}}_\mathbf{i}, \mathbf{y_i})$ w.r.t $\mathbf{W}$
    Update parameters $\mathbf{W^*} \leftarrow \mathbf{W} + \eta \frac{\partial \mathbf{C}}{\partial \mathbf{W}}$
return $\mathbf{W^*}$

The gradient of a neural network's parameters with respect to its cost function computed from a single observation can be noisy so a common alternative to SGD called minibatch SGD is often used. Minibatch SGD tends to produce smoother gradient updates at each step by computing the cost function and gradients based on a sample of m observations (m is often referred to as the "batch size") rather than a single datapoint as in SGD.

Algorithm 4 Minibatch Stochastic Gradient Descent for Training a Neural Network

Input: Training dataset of n input/output pairs $\{(\mathbf{x_1}, \mathbf{y_1}), \ldots, (\mathbf{x_n}, \mathbf{y_n})\}$
Input: Neural network model function $f(\mathbf{x}; W)$ parameterized by $\mathbf{W}$
Input: Cost function C
Input: Learning rate η
for $epoch$ in $1, 2, \ldots, epochs$ **do**
    Sample a minibatch of m training dataset observations $\{(\mathbf{x_1}, \mathbf{y_1}), \ldots, (\mathbf{x_m}, \mathbf{y_m})\}$
    Initialize gradient for this minibatch $\hat{\mathbf{g}} \leftarrow 0$
    for $i = 1$ to m **do**
        Compute current prediction using forward propagation $\hat{\mathbf{y}}_\mathbf{i} = f(\mathbf{x_i}; \mathbf{W})$
        Compute the cost $C(\hat{\mathbf{y}}_\mathbf{i}, \mathbf{y_i})$
        Update minibatch gradients using backpropagation $\hat{\mathbf{g}} \leftarrow \hat{\mathbf{g}} + \frac{1}{m} \frac{\partial \mathbf{C}}{\partial \mathbf{W}}$
    Update parameters $\mathbf{W^*} \leftarrow \mathbf{W} + \eta \frac{\partial \mathbf{C}}{\partial \mathbf{W}}$
return $\mathbf{W^*}$

The non-linearities in a neural network cause the cost function to be non-convex and therefore gradient descent with backpropagation is not guaranteed to find a global minimum of the network's cost function, only a local minimum which may not be the overall lowest cost possible for the model. This issue caused by the non-convexity if the cost function was long thought to be a major drawback of neural networks but in practice it is not. (LeCun, Bengio, and Hinton, 2015). One explanation due to Dauphin et al. (2014) for why local minima are not an issue is that in high dimensional space, stationary points in the error surface

are not local minima but saddle points since in high dimensional space there is almost certainly at least one dimension in which the cost surface can be reduced. It can be difficult for SGD as a first-order gradient descent algorithm to escape these seeming local minima and as such several second-order approaches have been proposed.

2.5.1 Learning rate adaptation and second-order gradient descent

The learning rate parameter controls the size of the steps taken during each step of the gradient descent process. Care has to be taken to not set the learning rate parameter too high which can cause the overall cost to increase or too low which can cause the model to take extremely long to converge. In practice, the learning rate parameter is typically set to a moderately high small number to start and reduced by a factor of 10 in a process called "learning rate annealing" after each time the network's cost converges. There are also several variations of the standard learning rate update procedure that tend to speed up and improve convergence during gradient descent.

The adaptive gradient (AdaGrad) algorithm (Duchi, Hazan, and Singer, 2011) stores the history of computed gradients between steps for each weight parameter and adapts the weight update to be larger for parameters that are updated less frequently. If $\mathbf{G_t}$ is a diagonal matrix with the sum of the squared gradients for each parameter up to the current update step at time t and ϵ is a small smoothing parameter to avoid division by 0, then the AdaGrad weight update rule is given by:

$$\mathbf{W^*} \leftarrow \mathbf{W} + \frac{\eta}{\sqrt{\mathbf{G}_t + \epsilon}} \frac{\partial \mathbf{C}}{\partial \mathbf{W}} \tag{2.19}$$

RMSProp (Tieleman and Hinton, 2012) is similar to AdaGrad is a second optimization order method that uses the second derivative of the cost function and divides the learning rate by a running average of squared gradients with decay parameter γ controlling the running average contribution to the updates:

$$E\Big[\frac{\partial \mathbf{C^2}}{\partial \mathbf{W}}\Big]_t = \gamma E\Big[\frac{\partial \mathbf{C^2}}{\partial \mathbf{W}}\Big]_{t-1} + (1-\gamma)\frac{\partial \mathbf{C^2}}{\partial \mathbf{W}} \tag{2.20}$$

$$\mathbf{W^*} \leftarrow \mathbf{W} + \frac{\eta}{\sqrt{E\Big[\frac{\partial \mathbf{C^2}}{\partial \mathbf{W}}\Big]_t + \epsilon}} \frac{\partial \mathbf{C}}{\partial \mathbf{W}} \tag{2.21}$$

Adaptive Moment Estimation (Adam) (Kingma and Ba, 2014) is another second-order optimization method also stores a running average of past squared gradient updates but extends these approaches by using both the first m_t and second v_t moments of the past

gradient updates for the current update step which are bias-corrected using decay rate parameters β_1 and β_2:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\frac{\partial \mathbf{C}}{\partial \mathbf{W}} \tag{2.22}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\frac{\partial \mathbf{C^2}}{\partial \mathbf{W}} \tag{2.23}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1} \tag{2.24}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2} \tag{2.25}$$

$$\mathbf{W^*} \leftarrow \mathbf{W} + \frac{\eta}{\sqrt{v_t + \epsilon}}\hat{m}_t \tag{2.26}$$

The Adam optimizer has been used to train several state-of-the-art models including SENet (Hu, Shen, and Sun, 2018) that won the ImageNet challenge in 2017.

2.5.2 Weight initialization

The standard approach to initializing the weights of a neural network until around 2010 was to draw random samples from a zero-mean Gaussian distribution with standard deviation 0.01 and initialize biases to 1 (Goodfellow, Bengio, and Courville, 2016). An alternative approach introduced by Glorot and Bengio (2010) named the *Xavier initialization* (after Glorot's first name) that tends to perform better and is more commonly used in practice suggests initializing the weight matrix $\mathbf{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ as:

$$\mathbf{W} \sim U\left[-\frac{\sqrt{6}}{\sqrt{d_{in} + d_{out}}}, +\frac{\sqrt{6}}{\sqrt{d_{in} + d_{out}}}\right] \tag{2.27}$$

where $U[a, b]$ is a uniformly sampled random variable in the range $[a, b]$ and d_in and d_out indicate the number of neurons connected to and from the given layer.

2.5.3 Training Protocols

When neural networks are used in supervised learning settings with datasets for which there are known ground-truth labels, the goal of training the network is not just for it to be accurate on the dataset on which it is trained but should generalize and be accurate on out-of-sample data too. A model with high accuracy predicting the in-sample training data but poor out-of-sample accuracy is said to have "overfit" the training data.

The standard approach to modeling using a neural network is to randomly split the dataset into training, validation and test splits (usually in 70%,20%,10% splits although this depends on the amount of data available). Sometimes cross-validation is used where the dataset is rotated through the splits in "folds" and the accuracy in each fold is averaged to

provide a better approximation of out-of-sample accuracy, especially in small datasets. The number of layers a network should have and the number of neurons that should be in each layer, together the learning rate and batch size are referred to as hyper-parameters. These hyper-parameters are selected by training many models on the training dataset and evaluating the trained model's performance on the validation set. After this hyper-parameter selection process is complete, the model is finally evaluated on the test set to obtain a true measure of how it will likely perform out-of-sample. A model that performs well on the training dataset but not on the validation dataset is said to have "over-fit" the training data and several "regularization" techniques have been developed to address this problem of over-fitting. The test set is set aside during the model development process and used only to obtain an objective assessment of the out-of-sample performance of the final model.

2.6 Regularization

Regularization is the term used for any technique that improves a machine learning model's ability to generalize to new data (Goodfellow, Bengio, and Courville, 2016). A large neural network trained for a long amount of time can learn very complex functions that may accurately fit the training data but not reflect the true underlying data generating distribution and therefore perform poorly on out-of-sample data. The techniques given below can be applied to any DNN - regularization achieved by more complex neural network architectures are given in subsequent chapters.

2.6.1 Early Stopping

At the start of the training procedure, a neural network's training and validation accuracy will likely increase but as training progresses, the network will start to over-fit the training data and the validation accuracy will begin to deteriorate. Early stopping stores a checkpoint of the model's weights after each epoch and if the validation accuracy does not improve for a number of successive epochs (called a *patience* parameter) then training is stopped and model weights from the best checkpoint before validation accuracy worsened are used.

2.6.2 Weight Regularization

The more weights a neural network has, the more capacity it has to learn complex functions that overfit the training data. Weight regularization puts constraints on the complexity of

the network by forcing its weights to take on small values which makes the distribution of the weights in the network more "regular" (Chollet, 2017). This is done by updating the model's cost function C to C^* to include additional regularization term that increases as the network's weights. The amount of regularization is controlled using a parameter λ chosen by cross-validation.

L1 Regularization adds a term proportional to L-1 norm of the weights:

$$C^* = C + \lambda \sum |w_i| \tag{2.28}$$

L2 Regularization adds a term proportional to L-2 norm of the weights:

$$C^* = C + \lambda \sum w_i^2 \tag{2.29}$$

These L-1 and L-2 norm regularization weight penalties are also used to control overfitting in Lasso and Ridge regression respectively (Friedman, Hastie, and Tibshirani, 2001). In the classification context with a categorical cross-entropy cost function the regularization term can be interpreted as coming from a Gaussian prior over the weight matrix $\mathbf{W}$ where instead of MLE the estimation process is performing Maximum a posteriori estimation (Goodfellow, Bengio, and Courville, 2016).

2.6.3 Dropout

Dropout is a form of regularization specific to neural networks invented by Hinton et al. (2012) that sets the output of each hidden neuron to zero with some probability at the start of each gradient descent step during training (but not during inference) The set of neurons "dropped-out" changes between training steps and dropout is only applied during training, not inference. Dropped-out neurons do not contribute to forward propagation and do not participate in back-propagation and as such, collections of neurons cannot co-adapt to memorize overly complex patterns in the training set, thus improving the generalization capacity of the network. Dropout increases the amount of time required to train neural networks. Dropout was cited by Krizhevsky, Sutskever, and Hinton (2012) as a critically important component of their ImageNet challenge winning architecture.

2.7 The Universal Approximation Theorem

The universal approximation theorem (Cybenko, 1989; Hornik, Stinchcombe, and White, 1989) states that a feedforward network with at least one hidden layer and any "squashing" activation function (such as the sigmoid or hyperbolic tangent) can approximate any

Borel measurable function from one finite-dimensional space to another with any desired non-zero error, provided that the network has enough hidden units. The concept of Borel measurability is beyond the scope of this work but essentially any function on a closed and bounded subset of $\mathbb{R}^n$ is Borel measurable and therefore can be approximated by a neural network (Goodfellow, Bengio, and Courville, 2016). Universal approximation theorems have also been proven for a wider class of activation functions including ReLU (Leshno et al., 1993).

The universal approximation theorem means that regardless of what function we are trying to learn (including functions that map video inputs to classification outputs), there exists a large feedforward neural network that will be able to *represent* this function. Unfortunately the theorem does not guarantee that the training algorithm used will be able to *learn* the function since the training algorithm might not be able to find the right parameters or may choose the wrong function due to overfitting. An analogy can be drawn to curve fitting where a polynomial with more parameters than datapoints can perfectly model training data but will likely not generalize to new data. On the other hand if the polynomial has very few parameters it may not have enough capacity to capture the regularity in the training data and extrapolate correctly to new data. The optimal choice is the minimum size model that represents the data and can generalize out-of-sample (LeCun et al., 1989a).

While in theory the universal approximation theorem states that there exists a network large enough to approximate a function with any desired level of accuracy, it does not specify how large this network will be and in the worst case an exponential number of hidden units may be required. In practice, more advanced DNN architectures such as Convolutional Neural Networks (discussed in Chapter 3) and Recurrent Neural Networks (discussed in Chapter 4) have been developed that take advantage of the structure of input data to impose constraints on network architecture and weights, thereby regularizing the network and making it possible to learn complex functions that generalize well with limited datasets and computational power.

Chapter 3

Deep Neural Network Architectures for Image Classification

This chapter begins by introducing image classification, the challenges involved and why fully connected neural networks are not a good solution. Section 3.2 presents the motivation, history and theory of Convolutional Neural Networks (CNNs). Section 3.3 introduces the ImageNet challenge and process of "transfer learning". Finally, Section 3.4 presents the CNN architectures that have won the ImageNet Classification Challenge since 2012.

3.1 Image Classification

Image classification refers to the task of assigning a single class to the main subject of an image and is different from object localization or instance segmentation tasks which seek to detect and locate the set of objects within an image.

Colour images are represented mathematically as 3-dimensional tensors with the first two dimensions representing width and height and the third dimension representing colour channels. While other colour space representations exist, images are usually stored using red, green and blue (RGB) colour intensity values for each discrete point in the image captured by the sensor in the camera (called a "pixel", a portmanteau of "picture element").

For example, a 224×224 pixel colour image would be represented using $224 \times 224 \times 3 = 150'528$ integers, each with a RGB colour intensity integer in $[0, 255]$ with $(0, 0, 0)$ corresponding to black, $(255, 255, 255)$ corresponding to white and other combinations of red/green/blue values able to represent any colour on the spectrum. A grayscale image is stored as a 2-dimensional tensor with width and height dimensions and grayscale intensity values in $[0, 255]$. At the time of writing, an entry level smartphone with a 10 megapixel camera can capture colour images with a pixel resolution of 4290×2800.

The high dimensionality of image data makes image classification a challenging modeling problem which is made more difficult by the high degree of variation inherent in images. In addition to intra-class variation where objects of the same class can contain significant natural visual variation (for example, different kinds of "dog" can look very different), there are many other sources of variation common in image data including:

Viewpoint variation	Objects viewed from different perspectives can appear different.
Scale variation	Objects appear bigger or smaller depending on how far away they are from the camera.
Pose variation	The same object can look different depending on its pose.
Illumination variation	Objects can look different depending on lighting conditions.
Background variation	Objects can appear in a range of settings with potentially noisy backgrounds and can blend into their environment.
Occlusion	Defining object features can be occluded by other objects nearer to the camera.

Traditional image classification approaches first extract lower dimensional discriminative features from the high dimensional image data and then use these features with a classification algorithm such as k-Nearest Neighbours (Keller, Gray, and Givens, 1985), Support Vector Machines (SVM) (Cortes and Vapnik, 1995) or a feedforward neural network to classify the image.

Principal Component Analysis (PCA) is an unsupervised dimensionality reduction technique that has been used with some success extract features for recognizing face images captured in a *controlled* environment ("eigenfaces") (Turk and Pentland, 1991) but PCA fails to outperform simply downsizing input images captured in *uncontrolled* settings where variation in the dataset is due not only to variation in the objects themselves but includes varying object positions, poses or backgrounds (LeCun, Huang, and Bottou, 2004). Several image-specific dimensionality reduction techniques such as Scale Invariant Feature Transform (SIFT) (Lowe, 1999) or Haar-like features (Lienhart and Maydt, 2002) have been developed that are robust to some sources of variance in image data. Until recently these kinds of feature extraction methods were a common pre-processing step in image classification but hand-engineered feature extraction techniques that encode expert knowledge do not generalize well and under-perform CNNs which jointly learn feature representations together with a classifier directly from raw image pixel data.

Fully-connected DNNs are not well suited to classifying images directly from image pixel data for 3 reasons:

- A single fully-connected layer with just 100 neurons connected to each colour channel of each pixel in a 224×224 colour input image would have $15'052'800$ trainable weight parameters which can cause computational challenges and likely cause the model to overfit its training data.
- Fully connected networks ignore the spatial topology of image data with pixels tending to be correlated with nearby pixels to form salient features within an image.
- Fully-connected networks have no invariance with respect to local distortions or translations causing them to fail on classification problems outside of controlled environments where objects out-of-sample often appear distorted or in different parts of an image than they did in training data.

In theory, due to the universal approximation theorem, a fully-connected network of sufficient size *could* learn to produce outputs that are translation invariant but this would require many units with identical weight patterns and an impractically large number of training examples to cover the space of possible variations (Bengio and Lecun, 1995). In practice, fully-connected DNNs applied to image classification tasks tend to overfit the training data due to their large number of parameters and do not generalize well out-of-sample (LeCun et al., 1989a).

3.2 Convolutional Neural Networks

Convolutional Neural Networks are a specialized kind of DNN architecture widely used for image classification (Goodfellow, Bengio, and Courville, 2016) that take advantage of the characteristics of image data to learn hierarchies of local translation invariant features that reduce the number of trainable parameters in the network while leaving enough representational power for CNNs to achieve state-of-the-art performance on image classification and other tasks.

CNNs were developed in the context of image classification but they have been found to be effective for many kinds of problems involving data that has a regular grid-like topology of locally correlated hierarchical features. They have been applied to 1-D data including time-series (LeCun, Bengio, et al., 1995) and sentence classification (Kim, 2014); 2-D data including speech recognition (Deng, Hinton, and Kingsbury, 2013); 3-D data including colour image classification and 4-D data including volumetric medical imaging scans (Milletari, Navab, and Ahmadi, 2016) and video classification (Yang et al., 2009).

3.2.1 Motivation

CNNs were inspired by visual perception in the biological mammalian brain with their connectivity patterns resembling the organization of the visual cortex. D. Hubel and T. Wiesel (1962) inserted a microelectrode into the visual cortex of an anesthetized cat and then projected light patterns onto a screen in front it. They found that individual cortical neurons in the cat's visual cortex were sensitive to specific patterns with certain neurons activating when shown vertical lines and others when shown diagonal or horizontal lines or light or dark patterns. Given fixed eye position, the region of visual space within which visual stimuli affect the firing of a single neuron is known as its *receptive field* and the experiments found that neighbouring neurons have similar and overlapping receptive fields. Later experiments on monkeys (Hubel and Wiesel, 1968) found that the size and location of receptive fields varied systematically across the cortex and are organized hierarchically to form a complete map of visual space from layers of simple lower level features.
These experiments in neurophysiology together with early work in pattern recognition that assembled neurons into two-dimensional arrays to extract local features (Fukushima, 1980) inspired the following three key architectural ideas used in CNNs (LeCun et al., 1989a):

Local Receptive Fields

Each convolutional unit receives input from a set of units located in a small neighbourhood of the previous layer, called the neuron's *receptive field*, which enables it to extract elementary local spatial features such as edges and corners.

Shared Weights

Each convolutional unit is scanned across the extent of the previous layer to compute an output plane called a *feature map* (see Figure 3.1). Since features that are useful in one part of an image are likely to be useful across the entire image, each convolutional unit is set to have identical weight vectors at each position that is scanned. This *weight sharing* introduces translation invariance and significantly reduces the number of trainable parameters in the network. A convolutional layer is composed of several convolutional units each computing a feature map so that different types of features are extracted at each location.

Subsampling

After a feature is extracted, its exact location becomes less important. Subsampling layers reduce the resolution of preceding feature maps while ensuring discriminative features persist through the network. This reduces the network's sensitivity to distortions in its inputs. Successive alternating convolutional and subsampling layers are stacked to reduce the spatial resolution of the network and extract higher level hierarchical features.

3.2.2 Convolutional Layers

A convolutional layer contains sets of convolutional units, also called *convolutional kernels*. Each convolutional unit is scanned across its input using the same set of weights in each position similar to a mathematical convolution operation, hence the name "convolutional layer". The first convolutional layer in a CNN scans across the input image but convolutional layers are usually stacked with later convolutional layers scanning across the outputs of earlier convolutional and subsampling layers (for example, see Figure 3.4).

A mathematical convolution essentially computes an arbitrary-dimensional weighted average with a weighting function being applied across the domain of an input function to produce an output. Convolutions are commonly used in image processing to produce filter effects, for example the 3×3 convolutional kernel illustrated in Figure 3.1 - called an "outline filter" in image processing software (Powell, 2017) - highlights edges in the input. In the image processing context, kernel weights are typically hand-chosen to produce a desired effect but in CNNs these kernel weights are randomly initialized and then *learned* using backpropagation so that each convolutional layer's output feature maps contain discriminative feature transformations that are useful for classification.

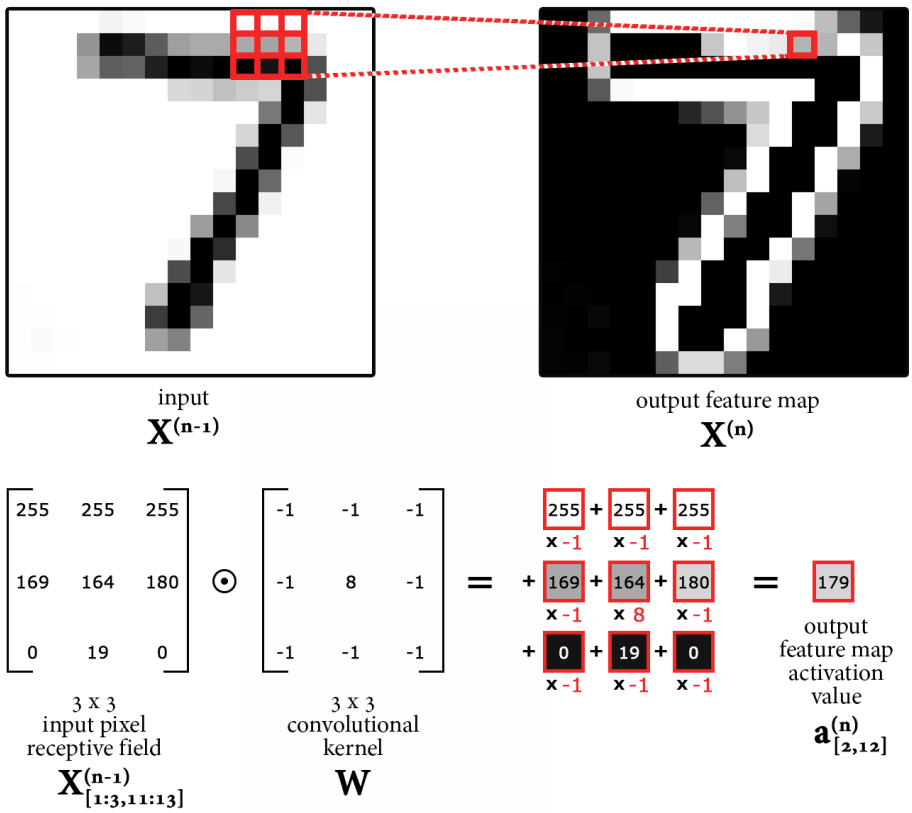


FIGURE 3.1: An example convolutional feature map computed by scanning a 3×3 convolutional unit (highlighted in red) across a 16×16 pixel grayscale input image (left) to produce output feature map (right). The calculation of the zoomed pixel highlighted in red on the output is shown beneath the images. The bias weight is set to zero and ReLU activation function is omitted for clarity. Black padding was used at the border to produce a valid convolution.

The number of pixels a convolutional kernel is moved as it scans across its input is referred to as its *stride*. The example above uses stride 1 but larger strides are also used to reduce the dimensionality of the output feature maps (Goodfellow, Bengio, and Courville, 2016). An *valid* convolution is one where a border of black or reflected pixels is added so that the convolution can be computed at the edges and produce an output feature map with the same dimensionality as its input.

Forward Propagation Through a Convolutional Layer

Forward propagation through a convolutional layer proceeds as illustrated above by scanning the convolutional kernel across each point in the input to compute a feature map output:

$$a_{(i,j)}^{(n,l)} = g(z_{(i,j)}^{(n,l)}) = g\left(\sum_{h,k,m} w_{(k,m)}^{(h,n,l)} a_{(i+k,j+m)}^{(h,l-1)} + w_{(0)}^{(n,l)}\right) \tag{3.1}$$

where:

$w_{(k,m)}^{(h,n,l)}$ = weight at position (k, m) connecting the h^{th} feature map of the $(l-1)^{th}$ layer to the n^{th} feature map of the l^{th} layer

$w_{(0)}^{(n,l)}$ = bias unit in the n^{th} feature map of the l^{th}layer

$a_{(i,j)}^{(n,l)}$ = output from unit in position (i, j) of the n^{th} feature map of the l^{th} layer

$z_{(i,j)}^{(n,l)}$ = pre-activation of the unit in position (i, j) of the n^{th} feature map of the l^{th} layer

$g(\cdot)$ = non-linear activation function applied element-wise

Backpropagation Through a Convolutional Layer

Backpropagation through a convolutional layer is similar to backpropagation in fully-connected layers but takes into account the local weight sharing property of convolution units so that gradients propagate through all weights in the kernel as it is applied to each position of its input:

$$\frac{\partial C}{\partial w_{(i,j)}^{(n,m,l)}} = \sum_{r,t}\left(\frac{\partial C}{\partial a_{(r,t)}^{(n,l)}}\frac{\partial a_{(r,t)}^{(n,l)}}{\partial w_{(i,j)}^{(n,m,l)}}\right) = \sum_{r,t}\left(\frac{\partial C}{\partial a_{(r,t)}^{(n,l)}} x_{(r+i,t+j)}^{(n-1,m)}\right) \tag{3.2}$$

3.2.3 Subsampling Layers

Subsampling in a CNN refers to the loss of some position information between layers as the input is transformed through the network, reducing the number of parameters in the network while enabling it to learn higher level hierarchical features. Subsampling layers follow convolutional layers in a CNN to increase the effective receptive field of its feature maps and reduce the resolution of successive convolutional layers, making the network more robust to shifts and distortions in the positions of objects in the input image (Bengio and Lecun, 1995). This helps the network generalize better since objects may appear at different positions out-of-sample than were present in the training data images.

Early CNNs used convolutions with stride greater than 1 to achieve subsampling (LeCun et al., 1989b) but later CNNs use explicit subsampling layers (Bengio and Lecun, 1995). A subsampling layer (also called a "pooling layer") is similar to a convolutional layer in that it consists of several subsampling units that each slide a window function over its input. Unlike a convolutional layer, the outputs of a subsampling layer have a lower dimensionality and the units in a subsampling layer typically do not have any trainable parameters therefore all the subsampling units in a subsampling layer are typically identical. Subsampling units typically use the same stride as window size so that they do not overlap. Two common types of functions used in pooling layers, max pooling and average pooling, use max and $average$ window functions respectively. Average pooling was the first subsampling operation used in CNNs (Bengio and Lecun, 1995) but max pooling is more commonly used in modern CNNs (Krizhevsky, Sutskever, and Hinton, 2012) and typically outperforms average pooling because averaging dilutes signal from previous layers (Scherer, Andreas, and Behnke, 2010).

The example below illustrates a max pooling operation with a 2×2 filter size and a window stride of 2 applied to a feature map to reduce its resolution by a factor of 4.

$$\begin{bmatrix} 1 & 0 & 1 & 2 \\ 0 & 0 & 1 & 3 \\ 3 & 3 & 6 & 3 \\ 1 & 2 & 4 & 7 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 3 \\ 3 & 7 \end{bmatrix}$$

Forward Propagation Through a Pooling Layer

Forward propagation through a pooling layer proceeds as illustrated in the example above by applying the subsampling operation to each point in the input feature map to compute an output feature map.

Backpropagation Through a Pooling Layer

In backpropagation, the backward pass for a max pooling unit only passes the gradient to the input from the previous layer that had the maximum value in the filter window from the forward pass (Goodfellow, Bengio, and Courville, 2016). Backpropagation through an average pooling unit averages the gradient in the subsequent layer into the corresponding inputs in the pooled layer.

3.2.4 Early Development of CNNs

CNNs were developed by LeCun et al. (1989a) while working on a handwritten digit recognition system to read postal codes using a dataset of 480 black & white 16×16 pixel images each containing a single handwritten digit.

FIGURE 3.2: Examples of handwritten digit images classified in LeCun et al. (1989a)

Five neural network architectures (shown in Figure 3.3) were compared:
The first two architectures were fully connected networks with one and two hidden layers ($2'570$ and $3'240$ weights) respectively. These were able to perfectly learn the training dataset but had "disappointing" generalization performance of 80% and 87% test accuracy respectively.
The third architecture used locally connected units to extract local features with the first hidden layer consisting of 8×8 units, each with a 3×3 pixel receptive field two pixels apart (stride 2). Each unit learned its own weights (no weight sharing). The second hidden layer used 4×4 locally connected units with a 5×5 receptive field. Although the network contained no explicit subsampling layer, the stride of 2 created two-to-one subsampling between layers. This locally connected network performed slightly better than the fully connected networks with 88.5% test accuracy but at much lower computational cost with just trainable $1'226$ weights.
Inspired by the visual cortex research described earlier, the fourth architecture introduced what would later become known as convolutional layers. The architecture was similar to the locally connected architecture but used *weight sharing* to detect the same features at different locations in the input while significantly reducing the number of trainable parameters in the network. The network featured two sets of feature maps in the first hidden layer, a locally connected second hidden layer with no weight sharing that were fully connected to the output layer. This network achieved significantly better test accuracy

of 94% with $2'266$ connections but only $1'132$ trainable weights.

Finally, the fifth network used a hierarchical structure that extended the weight sharing approach used in the fourth network to the second hidden layer, replacing the locally connected layer with four convolutional units which were fully connected to the output layer. This network performed the best with 98.4% test accuracy and $5'194$ connections but only $1'060$ trainable weights, showing that reducing the number of parameters in the network using the CNN architecture leads to improved generalization performance.

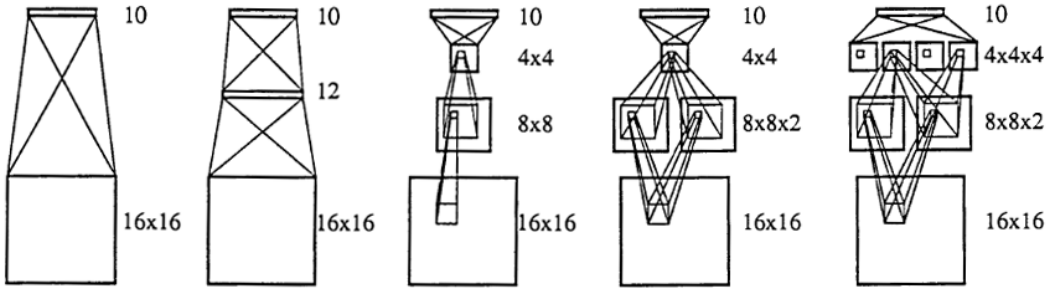


FIGURE 3.3: The five network architectures for digit recognition analyzed in LeCun et al. (1989a)

Architecture	Connections	Trainable Weights	Test Accuracy
1 layer fully connected network	2 570	2 570	80%
2 layer fully connected network	3 240	3 240	87%
3 layer locally connected network	1 226	1 226	88%
3 layer CNN with 1 conv. layer	2 266	1 132	94%
3 layer CNN with 2 conv. layers	5 194	1 060	98%

TABLE 3.1: Generalization performance for 5 network architectures analyzed in LeCun et al. (1989a). Performance on the training set is 100% for all networks.

A follow-up paper (LeCun et al., 1989b) built on the initial success of the first CNN architecture, using a larger dataset with $9'298$ images and significantly increasing the number of feature maps to 12 in each of the first two hidden layers. A test accuracy on this larger test set of 95% was reported. A year later, another CNN was trained (LeCun et al., 1990) on the same dataset that introduced an explicit subsampling layer using average pooling rather than stride for subsampling and achieved 3.4% test accuracy.

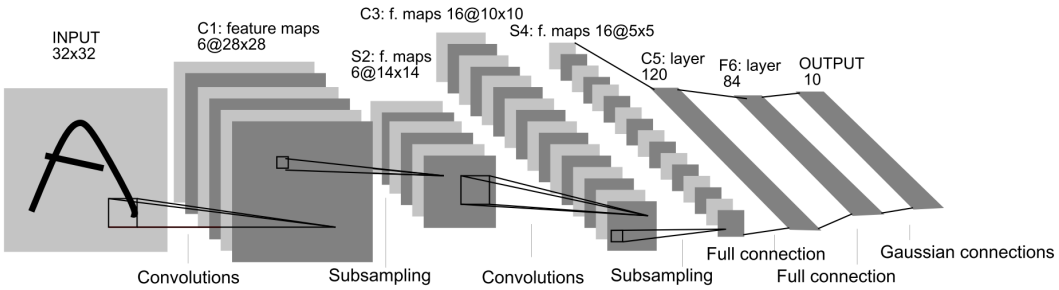


FIGURE 3.4: LeNet-5, a Convolutional Neural Network for hand-written digit recognition by LeCun et al. (1998). The input image is followed by two sets of alternating convolution ($C1$ & $C3$) and subsampling ($S2$ & $S4$) layers, with a square plane representing each separate feature map whose weights are identical. The fifth layer $C5$ is depicted as convolutional but acts as a fully connected dense layer into another dense layer $F6$ that is fully connected to the final 10-digit output classification layer.

Follow up work by Bengio and Lecun (1995) again increased the dataset size introducing the Modified National Institute of Standards and Technology (MNIST) database with $60'000$ training and $10'000$ test digit images written by 500 different writers. The digit images were resized to 28×28 pixels. The network trained on this dataset named LeNet-5 was deeper than previous models with 7 layers, again increasing the number of feature maps used in each layer and using alternating convolutional and subsampling layers for the first 4 hidden layers. The final 2 hidden layers in the network were fully connected to aggregate the features extracted in the previous spatial layers. The architecture (shown in Figure 3.4) achieved a test accuracy of 99.05%. A sensitivity analysis showed that halving the dataset size almost doubled the test error, highlighting the value of not only increased network depth but also increased training dataset size.

3.3 ImageNet

The ImageNet project is a large open database of annotated images that "aims to provide the most comprehensive and diverse coverage of the image world" (Deng et al., 2009). ImageNet is an ongoing project that aims to contain on the order of 50 million labeled full resolution images. At the time of writing, the ImageNet database contains $14'197'122$ images in $21'841$ hierarchical categories called "synsets" that map to the semantic hierarchy of WordNet ("synset" is a portmanteau of "synonym set"). The ImageNet database was constructed by first collecting candidate images using image search results and then cleaning the images using the Amazon Mechanical Turk crowdsourcing platform to obtain correct image labels.

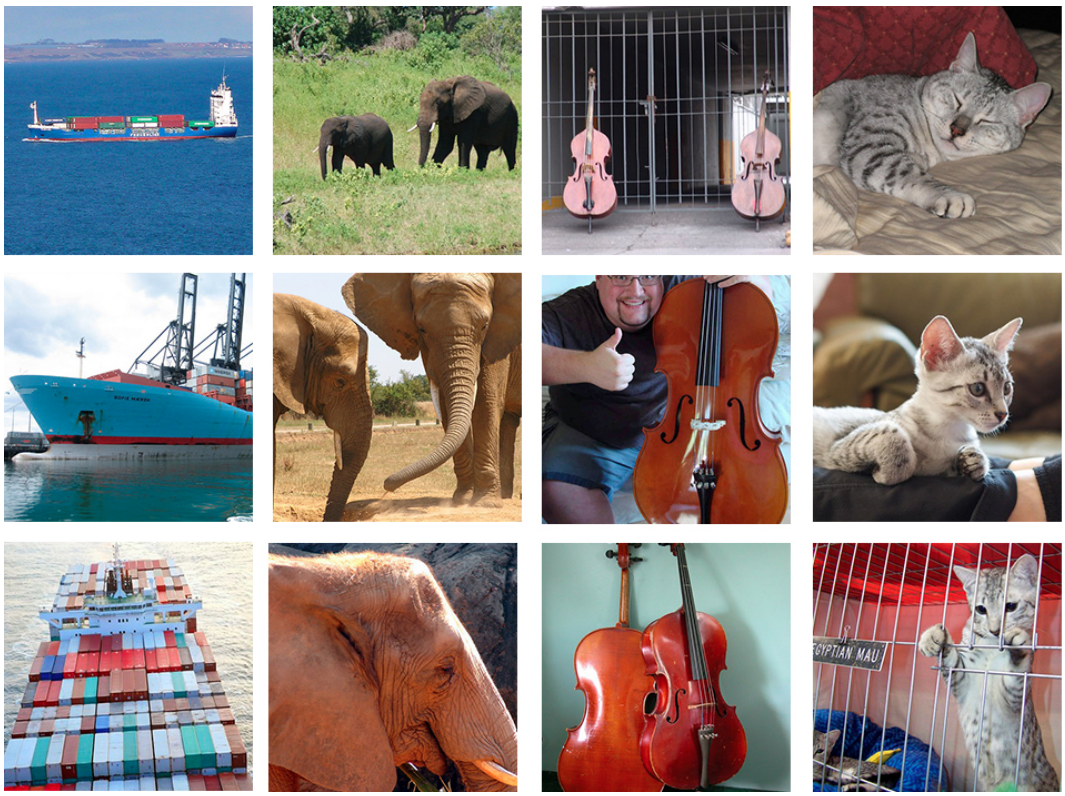


FIGURE 3.5: Images from the "Container Ship", "African Elephant", "Cello" and "Egyptian Cat" categories in the 1′000 category ImageNet ILSVRC subset

3.3.1 The ImageNet Large Scale Visual Recognition Challenge

The ImageNet project includes an annual challenge called the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) that is a benchmark of algorithms for object detection and image classification at large scale (Russakovsky, Deng, Su, Krause, Satheesh, Ma, Huang, Karpathy, Khosla, Bernstein, et al., 2015a). The dataset for the ILSVRC image classification challenge contains a subset of the full ImageNet database with 1′200′000 images across 1′000 diverse categories. The images in the ILSVRC dataset contain significant viewpoint, scale, pose, illumination and background variation as shown in Figure 3.5.

ILSVRC results

Algorithms competing in the ILSVRC image classification challenge need to submit 5 labels (ranked on confidence) for each image in the challenge test set. ILSVRC reports top-1 and top-5 error rates for each algorithm submission. A top-5 error indicates that none of the 5 labels submitted by an algorithm was correct. The table below shows the top-1 and top-5 error rates for the ImageNet challenge from 2010 to 2017.

Model	Year	Layers	Top-1 Accuracy	Top-5 Accuracy	Parameters
XRCE	2011	NA	NA	74.0%	NA
AlexNet	2012	8	62.5%	83.0%	60'000'000
ZFNet	2013	8	64.0%	85.5%	-
VGG16*	2013	23	71.3%	90.1%	138'000'000
GoogLeNet	2014	27	NA	93.3%	5'000'000
ResNet50	2015	168	74.9%	92.1%	25'600'000
InceptionV3*	2015	159	77.9%	93.7%	23'800'000
InceptionResNetV2*	2016	572	80.3%	95.3%	55'800'000

TABLE 3.2: ILSVRC winning architectures performance.
* 1st runner up

The significant 38% relative drop in top-5 error rates between the winning models in 2011 and 2012 marks the first submission of a deep CNN to the challenge. The 2011 model used Fischer vectors and SIFT features with a SVM classifier. Every ILSVRC winning model since 2012 has used a deep CNN with the decreasing error rates associated with increasingly deeper CNNs. While the CNN architecture was developed in LeCun et al. (1989a) and successfully applied to small scale image recognition problems such as MNIST, improvements in GPU hardware and the introduction of non-saturating ReLU activation functions made it possible to apply CNNs to large scale image classification tasks. The 2016 and 2017 winning models omitted because they were ensembles of several CNNs. Two additional single networks that have performed well since 2015 are included as their architectures will be discussed later. Human top-5 ILSVRC error was found to be 5.1% (Karpathy, 2014).

3.3.2 Transfer Learning

Transfer learning refers to the use of a model that has been learned on one dataset to improve performance on another dataset (Goodfellow, Bengio, and Courville, 2016). The low-level visual features such as edges, shapes, lighting, textures, and so on learned on one image classification task are likely to be useful for another even if the classes being predicted are not the same.

Since CNNs are feedforward networks that transform input image data into intermediate feature representations and ultimately class prediction outputs, one approach to transfer learning is to truncate a "pretrained" CNN learned on another dataset at an intermediate layer and use the truncated network a feature extractor. New layers can be added to the truncated network and the weights of those additional layers can be learned on the new

dataset to learn a new classifier. Alternatively, the output layer of the pre-trained CNN can be replaced for a new task with new classifier weights trained while also updating some or all of the weights of the feature extractor network in a process called "fine-tuning".
Most of the ILSVRC winning architectures presented in this chapter are open source and weights trained on ImageNet are freely available for download. This has enabled successful results in classifying images in small datasets since considerably less data is required to "fine-tune" a CNN that has already learned features than is required to learn the features of a CNN from scratch.
Fine-tuning has been used successfully in ecology by Gomez Villa, Salazar, and Vargas (2017) to classify animals captured in a camera trap in the Serengeti into one of 26 classes with 88.9% top-1 and 98.1% top-5 test set accuracy transfer learning from a pretrained ResNet CNN. Similarly (Weinstein, 2018) fine-tunes InceptionV3 with approximately 30 thousand images to classify whether a hummingbird is present in camera-trap images achieving a 89.3% true positive rate and (Siddiqui et al., 2018) fine-tune AlexNet, VGGNet and ResNet to classify fish underwater using only approximately 100 training images per class achieving 89% accuracy transfer learning from ResNet.

3.4 Architectures for Image Classification in ILSVRC

This section presents the architectures that won the ILSVRC image classification challenge or were first runner-up from 2012 to 2015. These architectures are presented so that their usefulness as pre-trained feature extractors for video classification in ecology can be investigated. As such, the winning architectures from 2016 and 2017 which were ensembles (that are not publicly available) rather than individual CNNs are not considered here. Another architecture that has performed well since 2016 and whose ImageNet trained weights are publicly available, InceptionResNetV2 is also considered.

3.4.1 Alex Net

AlexNet (Krizhevsky, Sutskever, and Hinton, 2012) was a breakthrough in deep learning which substantially reduced the ILSVRC classification previous year's top-5 test error rate from 26% to 17%, a relative 34.6% reduction in error. It was the first DNN to compete in ImageNet.
AlexNet has the same essential structure as LeNet-5 with alternating convolutional and

subsampling layers. Instead of receiving 28×28 pixel inputs, it receives colour images resized to 224×224 pixels. It has 8 layers. The first 5 layers are convolutional with increasing numbers of feature maps of decreasing receptive fields (from $11 \times 11 \times 3$ in the first layer to $3 \times 3 \times 192$ in the fifth layer, stride 4 in the first layer and stride 1 in subsequent layers). AlexNet used 3×3 max pooling in the first two layers. Finally they used 2 dense layers before their 1000 class classification layer.

AlexNet used ReLU activations which were noted as important to its success due to their speed and its implications on their ability to iterate architectures. It was trained with dropout, the authors remarking that "without dropout, our network exhibits substantial overfitting". The network had $60'000'000$ trainable parameters and had to be split across two high performance GPUs as shown in Figure 3.6. Training took five to six days on two 3GB GPUs.

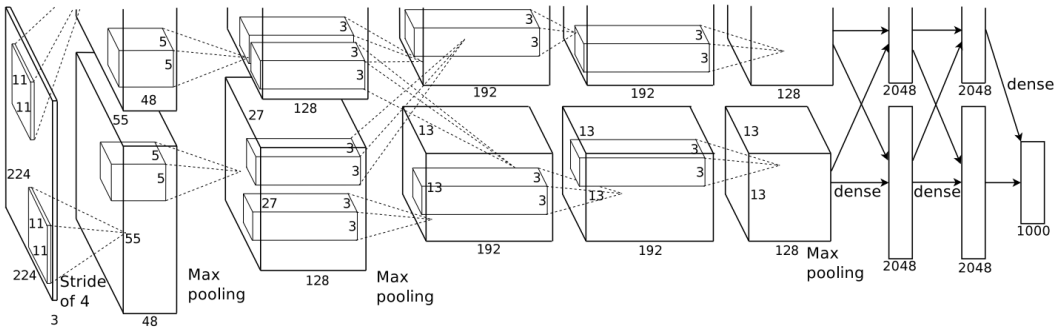


FIGURE 3.6: Architecture of the AlexNet (Krizhevsky, Sutskever, and Hinton, 2012) Convolutional Neural Network that won the ILSVRC competition in 2012. The model was spread across two GPUs by placing half the kernels on each GPU and limiting cross-GPU communication to certain layers because the memory limits of available GPUs at the time meant that the entire model could not be trained on a single GPU.

3.4.2 ZF Net

ZFNet (Zeiler and Fergus, 2013) built on the success of AlexNet through careful analysis of its architecture using a new technique they developed called *deconvolution* which essentially reverses the effects of a convolution kernel enabling them to visualize each feature in the pixel domain. These visualizations are shown in Figure 3.7 and show how CNNs learn higher level features deeper in the network by building hierarchies from combinations of lower level features. The visualizations also showed extremely high and low frequencies in the first convolutional layer and an aliasing problem int he second. These insights were used to inform modifications to AlexNet including reducing the first layer kernel size to 7×7 and reducing its stride to 2 to capture more mid frequencies, leading to slight improvements and winning ILSVRC in 2013.

FIGURE 3.7: The top 9 activations in a random subset of feature maps from ZFNet

3.4.3 VGG Net

VGGNet (Simonyan and Zisserman, 2014) was the first runner-up in ILSVRC 2014. The VGGNet architecture (see Figure 3.8) differs from AlexNet and ZFNet in that it uses the same sized 3×3 overlapping convolutions throughout the network. By stacking successive layers of 3×3 convolutions followed by pooling layers, the effective receptive field increases between layers while reducing the number of parameters in the network. It

was also much deeper than the previous ImageNet CNNs with 23 layers.

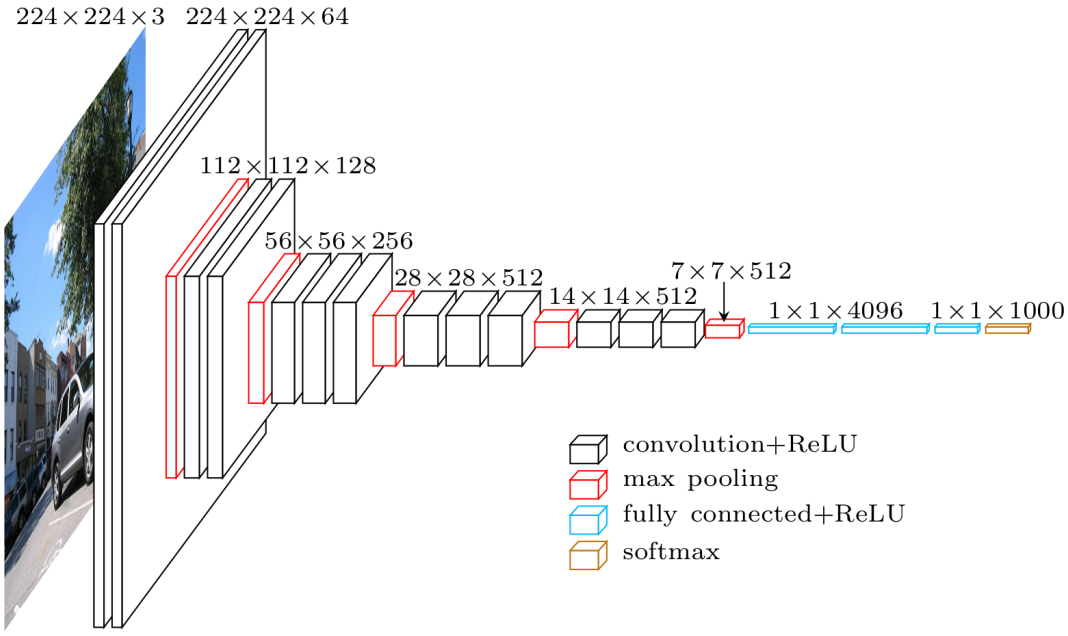


FIGURE 3.8: Architecture of VGGNet (Simonyan and Zisserman, 2014), a Convolutional Neural Network that placed second in the ILSVRC competition in 2014.

3.4.4 GoogLeNet / Inception v1

GoogLeNet, also called Inceptionv1 (Szegedy, Liu, et al., 2015; Ioffe and Szegedy, 2015), won ImageNet in 2014. It featured a new way of applying convolutional units called an "inception module" shown in Figure 3.9 which combines convolutions of different sizes as shown in Figure A.2 to extract features at different levels of magnitude at each layer. These inception modules aggregate information throughout the network mitigating the need for large dense layers at the end of the network, reducing the total number of parameters and enabling a much deeper architecture without overfitting. The Inceptionv1 architecture (see full architecture in Appendix B) is 27 layers deep.

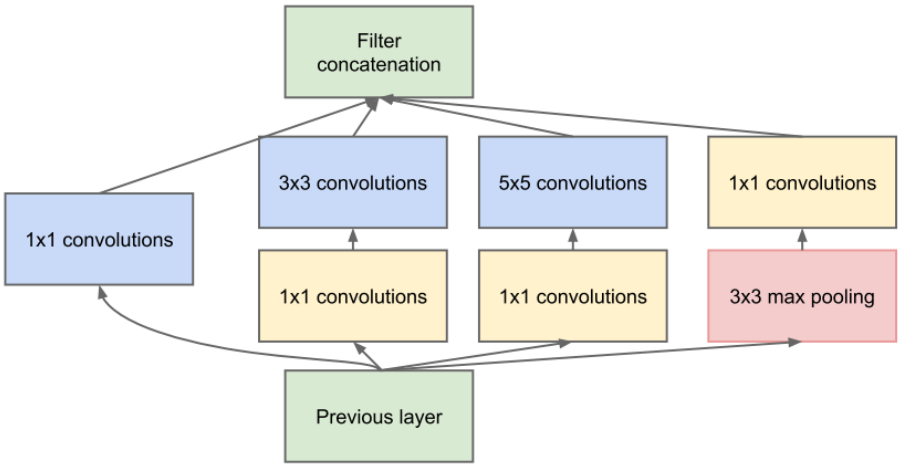


FIGURE 3.9: The Inception Module (with dimensionality reduction) from GoogLeNet (()Szegedy2015)

3.4.5 ResNet

ResNet (He et al., 2015) won ImageNet in 2015. It introduces residual skip connections shown in Figure 3.10 that alleviate the "vanishing gradient problem" because they allow the gradient to flow via the skip connection regardless of the gradient flowing through the branch. It is much deeper at 168 layers (see full architecture in Appendix B).

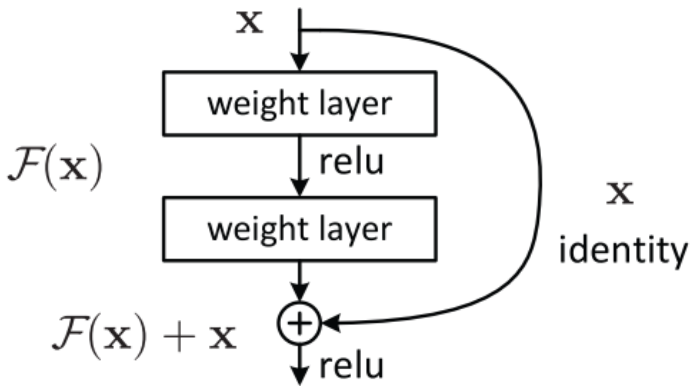


FIGURE 3.10: A residual connection from ResNet (He et al., 2015)

3.4.6 Inception v2, Inception v3 & Inception-ResNet-V2

Inception v2 and v3 did not win ImageNet but further developed the Inception module to achieve even better results in a single CNN architecture (Szegedy, Vanhoucke, et al., 2015). It did this by factorizing the Inception module as shown in Figure 3.11. Inception-ResNet-V2 (Szegedy et al., 2017) similarly did not win ImageNet but further developed the

factorized Inception module approach by adding a residual connection between the base and output nodes of the Inception module.

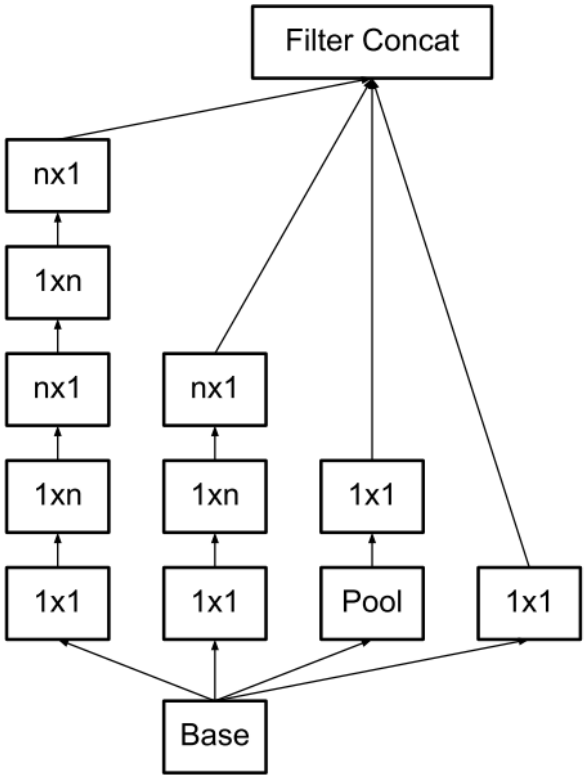


FIGURE 3.11: The Factorized Inception Module from Inception v2 and v3 (Szegedy, Vanhoucke, et al., 2015)

The effectiveness of these incremental CNN architecture improvements in the context of video classification for ecology will be investigated after first introducing how DNNs can be applied to video classification in the following chapter.

Chapter 4

Deep Neural Network Architectures for Video Classification

This chapter begins by introducing video classification, the challenges involved and various approaches used including traditional and DNN models. Section 4.2 introduces spatio-temporal CNN architectures. Section 4.3 presents the theory and motivation of Recurrent Neural Networks (RNNs). Section 4.4 introduces spatial-then-temporal DNN models. Finally, Section **??** surveys DNNs for video classification in Ecology.

4.1 Video Classification

A video is a series of images called "frames" which appear to be moving when shown quickly in sequence. The videos shown in cinemas are recorded at a *frame rate* of 24 frames per second (FPS). Video classification is the task of assigning a single class label to an entire video, part of a video (a "video clip") or a sequence of individual video frames. Video classification benchmark datasets (for example UCF-101 shown in Figure 4.1) typically involve recognizing actions in videos so frame-level video classification is called "temporal action localization" (Shou, Wang, and Chang, 2016). While videos are often accompanied by audio, the video classification task excludes audio and is limited to visual data.

Colour videos are represented mathematically as 4-dimensional tensors with the first three dimensions representing each colour frame (width, height, RGB) and the fourth dimension representing the sequence of frames over time. For example, a 60 minute 224×224 pixel colour video recorded at 24 FPS can be represented as a $[224, 224, 3, 60 \times 60 \times 24]$ tensor using $224 \times 224 \times 3 \times 60 \times 60 \times 24 = 13'005'619'200$ integers in $[0, 255]$.

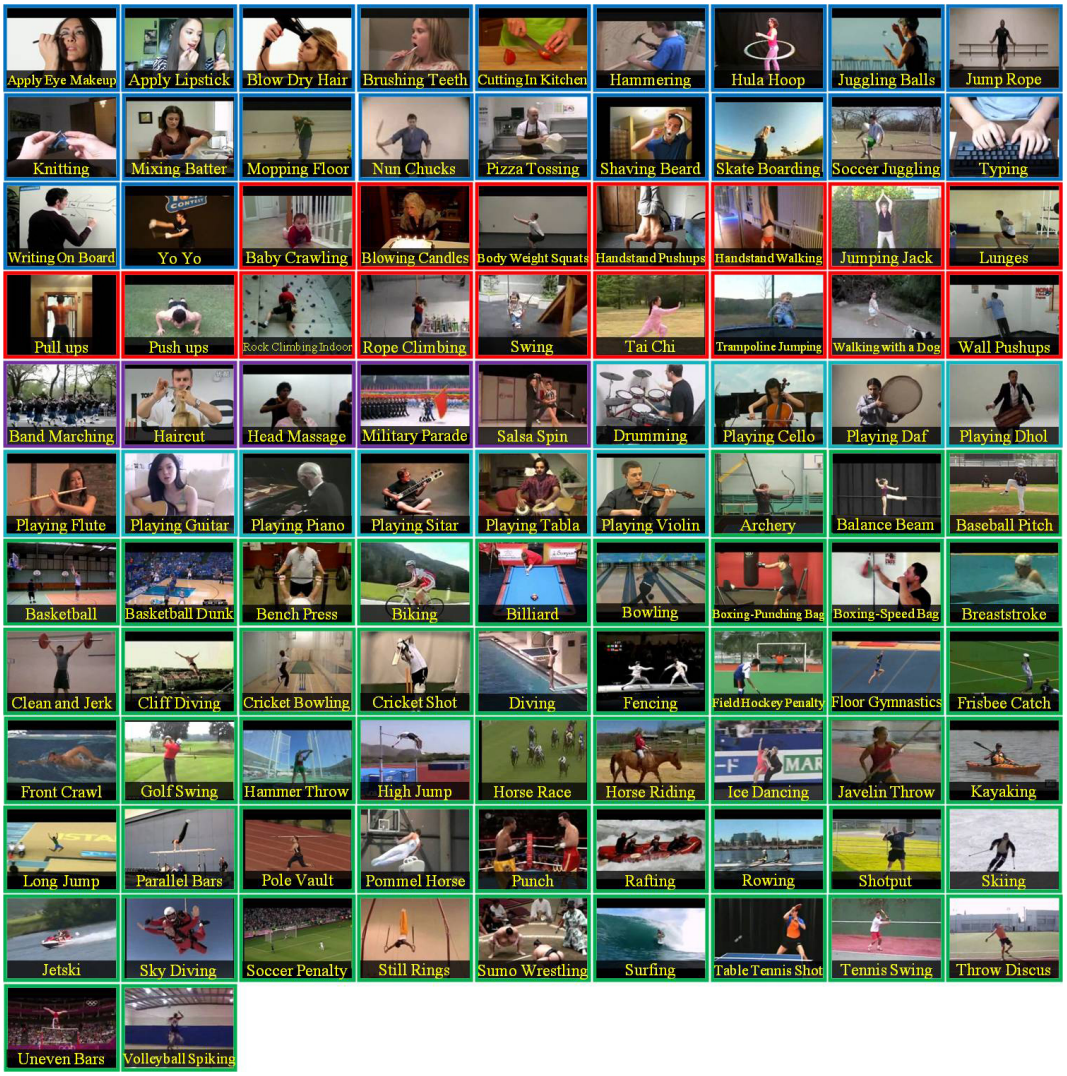


FIGURE 4.1: The 101 action categories in the UCF101 "Human Actions Classes from Videos in the Wild" dataset (Soomro, Zamir, and Shah, 2012) containing 13'320 videos recorded at 25 FPS with a mean clip length of 7.2 seconds. The action classes are grouped into five categories highlighted in blue, red, purple, teal and green respecitvely: Human-Object Interaction, Body-Motion Only, Human-Human Interaction, Playing Musical Instruments, Sports.

Video classification is a very challenging modeling problem with the challenges of image classification amplified because the same sources of natural visual variation occur not only between videos but also within videos as objects move around and change poses, scales, illuminations and sometimes backgrounds during the course of a single video. The video camera itself can move around during recording, introducing more visual variation especially in an ecological context where cameras move due to wind or are mounted to underwater fishing nets on moving vessels or cameras are attached to animals moving around their environment. The temporal component of video also presents significant modeling challenges not only because it dramatically increases the size of video data but because the relevant visual features required to classify a video can span several frames with no single frame containing enough information on its own. The pixels of an image representing objects are not only correlated spatially to form visual object features in a single frame but are also correlated through time (Krizhevsky, Sutskever, and Hinton, 2012).

Traditional approaches to video classification are similar to image classification. First lower-dimensional features are extracted using hand-engineered feature extraction algorithms (e.g. SIFT or Harris features) then these are input into a classifier. The simplest approach ignores the motion information contained in the temporal ordering of frames and treats video classification as a set of image classification tasks together with a majority voting algorithm to combine frame-level classification outputs into a video classification. Several hand-engineered spatio-temporal feature extraction algorithms such as the Cuboid detector (Dollár et al., 2005) and Harris-3D detector (Laptev et al., 2008) have been developed to capture motion information between frames but these require the features to be known and fixed in advance. Two broad classes of approaches to DNNs to video classification include:

1. Spatio-temporal Spatial and temporal features are extracted simultaneously by extending convolutions through time

2. Spatial-then-temporal Spatial features are extracted from frames (usually using a CNN) which are combined temporally using pooling or a RNN

4.2 Spatio-temporal CNNs

CNNs can be modified to incorporate motion information in videos by extending their convolution from 2 spatial dimensions (width and height) to 3 spatio-temporal dimensions (width, height and time). Like a 2D convolutional kernel is scanned over the width and height of an image, a 3D convolutional kernel is scanned over the width, height and temporal dimension of multiple contiguous frames stacked together. The forward and back-propagation equations for a 3D convolution are the same as equations 3.1 and 3.2 but with an additional summation index for the temporal dimension of the 3D convolution.

The first "3D CNN" model was developed by Ji et al. (2010) to classify actions in the TRECVID video surveillance dataset consisting of short clips of people taking one of three actions ("celltoear", "objectput", "pointing") captured at London Gatwick airport.

The model (shown in Figure 4.2) consists of a hardwired layer and six trainable layers. It takes as input 60×40 pixel video clips of 7 frames cropped to a single person. The first layer performs hardwired transformations to compute five different channels denoted by gray, gradient-x, gradient-y, optflow-x and optflow-y. The gradient channels compute the pixel-wise gradient at each spatial point within frames and the optflow channels compute optical flow (Horn and Schunck, 1981) between frames which is a measure of relative motion. In the second layer (labelled *C2*), these five channels (analogous to RGB colour channels) are convolved separately by a 3D convolution with kernel size

of $7 \times 7 \times 3$ (7×7 in the spatial dimension and 3 in the temporal dimension) and two sets of convolutions are applied at each location resulting in two sets of feature maps. A subsampling layer *S3* applies 2×2 subsampling on each feature map to reduce the model's spatial resolution, and is followed by another 3D convolutional layer *C4* with 3 different 3D kernels of size of $7 \times 6 \times 3$. Another subsampling layer *S5* applies 3×3 subsampling and the final convolutional layer *C6* is a 7×4 spatial convolution that produces a 128 neurons which are fully connected to the final 3-class output layer. The 3D CNN model achieved state-of-the-art accuracy on TRECVID, outperforming a 2-D CNN baseline.

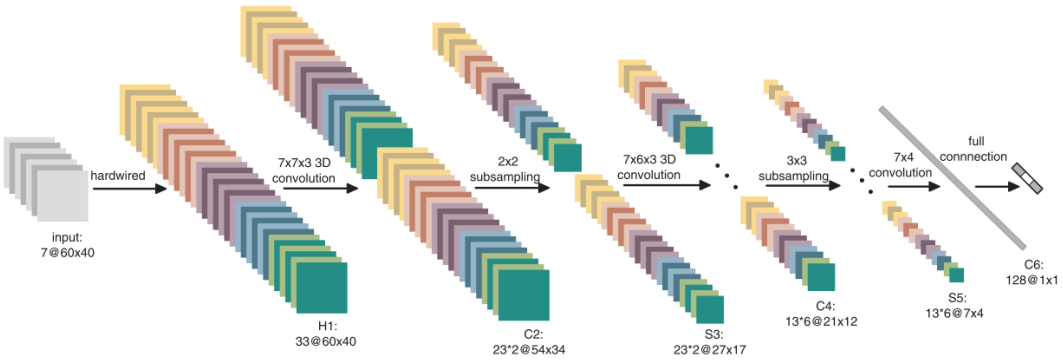


FIGURE 4.2: Architecture of the 3-D CNN (Ji et al., 2010) for video activity recognition

Following the success of CNNs in the ILSVRC challenge in 2013, Karpathy et al. (2014) investigated different approaches to temporally fusing the connectivity of the ImageNet-winning AlexNet CNN across frames to take advantage of local motion information for video classification on the Sports-1M and UCF101 datasets which contain much longer videos with many more classes than TRECVID. Three approaches were investigated:

Late Fusion Merges two single-frame AlexNet CNNs with shared parameters 15 frames apart at the first fully-connected layer. Neither single-frame network alone can detect motion but global motion characteristics can be derived through the combined outputs of each frame network.

Early Fusion Extends the first convolutional layer of AlexNet to have an input of 10 frames and a spatio-temporal kernel with size $11 \times 11 \times 3 \times 10$. Motion information across frames is merged at the start of the network followed by alternating subsampling and convolutional layers.

Slow Fusion Extends the connectivity of all convolutional layers in time. The network input is a 10 frame clip. The first convolutional kernel has size $11 \times 11 \times 3 \times 4$ with stride 2 to produce 4 responses in time and the later convolutional layers repeat this pattern so that the third convolutional layer's receptive field includes all 10 input frames.

Early Fusion was found to perform the best, followed by Late Fusion, Slow Fusion, and then a single frame CNN baseline. Since each network received at most 10 frames as input, video-level predictions were produced by presenting 20 randomly sampled clips from each video in the dataset to the model and then averaging the clip predictions. Although a similar architecture to the 2013 ILSVRC winning CNN was used, it was trained from scratch and Karpathy et al. (2014) did not use transfer learning from ImageNet.

Many different 3D CNN architectures were analyzed by Tran et al. (2015) across a range of video classification datasets with longer videos and more varied classes than TRECVID and the architecture shown in Figure 4.3 was found to perform the best. It has 8 convolution layers, 5 max pooling layers and 2 fully connected layers. All convolution kernels are $3 \times 3 \times 3$ kernels with stride 1 in both spatial and temporal dimensions. This architecture, called "C3D", consistently outperformed a 2D CNN baseline across datasets. It achieved state-of-the-art performance on UCF-101 and was competitive with the best model on Sport1M at the time called "DeepVideo". Unlike spatio-temporal CNN models, DeepVideo follows the spatial-then-temporal approach to extending CNNs for video applications by using recurrent neural network architectures introduced in the following section.

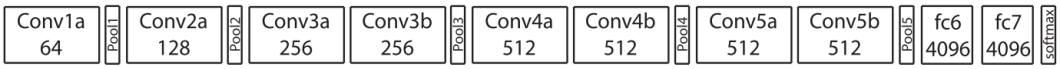


FIGURE 4.3: Architecture of the C3D (Tran et al., 2015) network for video activity recognition

4.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a specialized kind of neural network used for processing sequential data such as video frames. CNNs can be used to model sequence data but the *same* convolutional kernel weights are convolved along all points in the input sequence and they can only model fixed-length sequences. In contrast, the weights in a recurrent unit are updated sequentially as each element of the sequence is processed by the network (Nielsen, 2015) and RNNs can model variable length sequences. Unlike a feedforward or convolutional neural network which can be represented as a directed acyclic graph, a RNN contains at least one feedback connection that creates a recurrent loop. This recurrence makes it possible for information to persist between elements of a sequence and between observations in a dataset, making RNNs powerful tools for sequence modeling problems. RNNs have been applied to classifying video (Donahue et al., 2014), text (Bahdanau, Cho, and Bengio, 2014), speech (Graves, Mohamed, and Hinton, 2013) and many other sequential datasets (Goodfellow, Bengio, and Courville, 2016).

There are several variations of Recurrent Neural Networks. The next subsection introduces the traditional RNN model and some of its drawbacks that have motivated the development of two popular variations discussed in the following subsections: Long-Short Term Memory Networks and Gated Recurrent Units. This section closes with a brief discussion of "stacked" RNNs.

4.3.1 Simple Recurrent Neural Networks

The traditional "simple" RNN (shown in Figure 4.4) models temporal dynamics by learning weights that map sequences of inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$ to hidden states $\mathbf{h}_t$ and hidden states to outputs z_t at each step t in the sequence. In a RNN, the hidden state $\mathbf{h}_t$ depends not only on the current observation $\mathbf{x}_t$ but also on the previous hidden state $\mathbf{h}_{t-1}$, thus $\mathbf{h}_t$ contains information about the whole sequence and acts as a sort of memory that captures dependencies in the sequence (Goodfellow, Bengio, and Courville, 2016).
The following recursive equations define a RNN unit:

$$\mathbf{h}_t = g(W_{xh}x_t + W_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{4.1}$$

$$\mathbf{z}_t = g(W_{hz}\mathbf{h}_t + \mathbf{b}_z) \tag{4.2}$$

where:

$$\begin{aligned}
\mathbf{x}_t &= \text{input sequence value at time } t \text{ with } t \in [0, T] \\
\mathbf{h}_t &= \text{hidden state at time } t \text{ with } N \text{ hidden units } (h_t \in \mathbb{R}^N) \\
\mathbf{z}_t &= \text{the RNN output at time } t \\
\mathbf{W}_{xh} &= \text{matrix of weights connected from the sequence input to the hidden units} \\
\mathbf{W}_{hh} &= \text{matrix of weights connected from the hidden units from } t-1 \text{ to } t \\
\mathbf{W}_{hz} &= \text{matrix of weights connected from the hidden to output units} \\
\mathbf{b}_h &= \text{vector of hidden unit biases} \\
\mathbf{b}_z &= \text{vector of output unit biases} \\
g(\cdot) &= \text{non-linear activation function (usually tanh) applied element-wise}
\end{aligned}$$

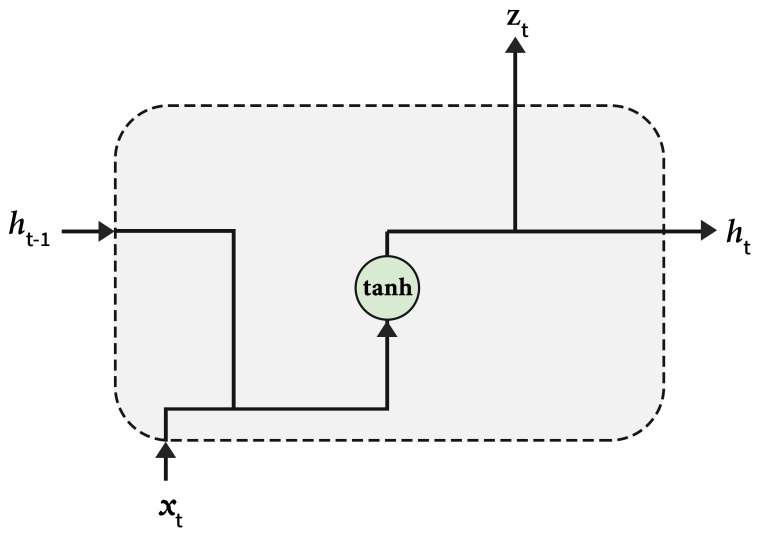


FIGURE 4.4: A recurrent unit, diagram adapted from Olah (2015)

Forward Propagation Through a RNN

Forward propagating information through an RNN involves sequentially passing elements of the input sequence to the RNN, computing the hidden state and then passing this hidden state to the computation in the next step as follows:

$$z_j^{l,t} = \sum_{i=0}^{d^{l-1}} w_{ij}^l x_i^{l-1,t} + \sum_{h=0}^{d^l} w_{hj}^l x_h^{l,t-1} \tag{4.3}$$

$$x_j^{l,t} = g\left(z_j^{l,t}\right) \tag{4.4}$$

where:

$d^l =$ the dimensionality of (number of neurons in) layer l

$x_i^{l,t} =$ the input in the i^{th} neuron in the l^{th} layer at time t

$z_j^{l,t} =$ the input to the jth neuron in the l^{th} layer at time t

$w_{ij} =$ weight from the i^{th} neuron of the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer

$g(\cdot) =$ non-linear activation function applied element-wise

Backpropagation Through Time

The algorithm used to train a RNN is called back-propagation through time or BPTT (Goodfellow, Bengio, and Courville, 2016). Like backpropagation for feedforward networks, BPTT propagates model errors from the network's outputs back through its hidden layers. The key difference in BPTT is that the network's output is a function not only of the outputs at the current time step but of all previous timestep outputs too. This requires errors to be propagated not only at the end but through all time steps in the sequence leading to the name back-propagation through time (Bishop, 2006).

Training an RNN using BPTT begins with unrolling the network in time into a larger feedforward network with T sub-networks laid out in a sequence and shared weight parameters in each. Then the backpropagation algorithm is used to compute the gradient of overall unrolled network cost with respect to the shared parameters in each of the sub-networks, starting at the end of the sequence T and moving back to the start of the sequence.

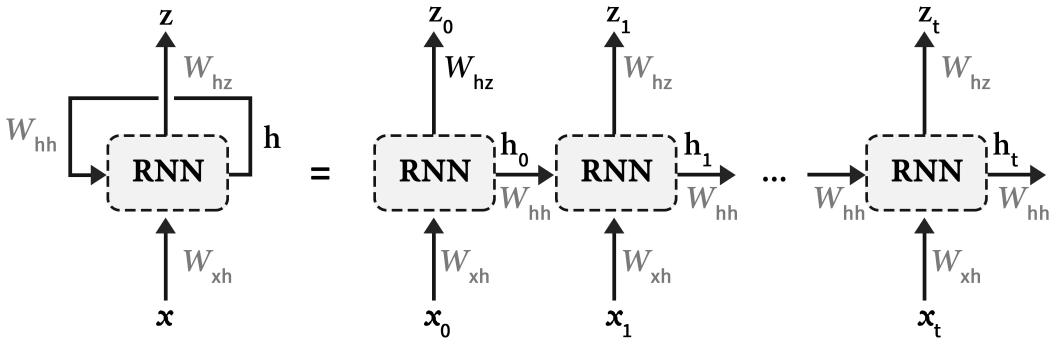


FIGURE 4.5: An unrolled recurrent neural network, diagram adapted from Olah (2015)

The Long-Term Dependency Problem

Many sequence modeling problems exhibit long-term dependencies where information critical to making a prediction at a point in time is located far back in the history of the sequence, for example forecasting daily time-series that exhibit annual seasonality or classifying actions in long videos where frames at the start of the clip contain information essential for classifying the video at the end. RNNs are in theory capable of solving such problems but in practice are unable to handle long-term dependencies due to two problems.

The primary problem is that gradients propagated over many steps tend to either "vanish" or "explode" (Bengio, Simard, and Frasconi, 1994). Unlike feedforward networks that have different weight matrices between layers, RNNs use the same weight matrix at each step when unrolled through time. When small gradient updates are propagated over many layers, the multiplicative effect causes them to shrink towards zero and "vanish" (and conversely for large gradients which "explode"), preventing the RNN from learning effectively. The second problem is that even if the parameters are such that the RNN is stable and gradients neither vanish nor explode, exponentially smaller weights are given to long-term interactions compared to short term interactions causing the model to perform poorly with long-term dependencies.

4.3.2 Long-Short Term Memory Units

The Long-Short Term Memory Unit are a type of RNN unit that was introduced by Hochreiter and Schmidhuber (1997) to address the long-term dependency problem. The key improvement in LSTM units is the addition of a self-loop path controlled by 3 trainable gating functions to control how gradients can flow through the unit. These allow the time scales at which information is integrated into the network to vary based on the input sequence. The gating functions control how present information and previous memory are used to update the current activation and produce the next output, mitigating the vanishing gradient problem.

A gate is just the output of one of the functions below which weights its inputs. For input dimension N, the gates used in an LSTM are called the input gate $i_t \in \mathbb{R}^N$, the forget gate $f_t \in \mathbb{R}^N$, and the output gate $o_t \in \mathbb{R}^N$. These give the LSTM the ability to add or remove information to its internal hidden state depending on the data and regulate the information that flows through the network thus enabling a LSTM unit to learn long term dependencies.

The LSTM unit is defined as follows with sigmoid function indicated by σ:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
\tilde{c}_t &= \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

The memory cell unit c_t combines the previous memory cell unit value c_{t-1} (modulated by f_t) with $\tilde{c}_t$, a function of the current input and previous hidden state (modulated by the input gate i_t). The input gate i_t and forget gates f_t are sigmoidal (with values in $[0, 1]$) and essentially scale the extent to which the LSTM will selectively forget its previous memory or incorporate its current input. Similarly the output gate o_t modulates how much of the memory cell value to transfer to the hidden state at a given time-step.

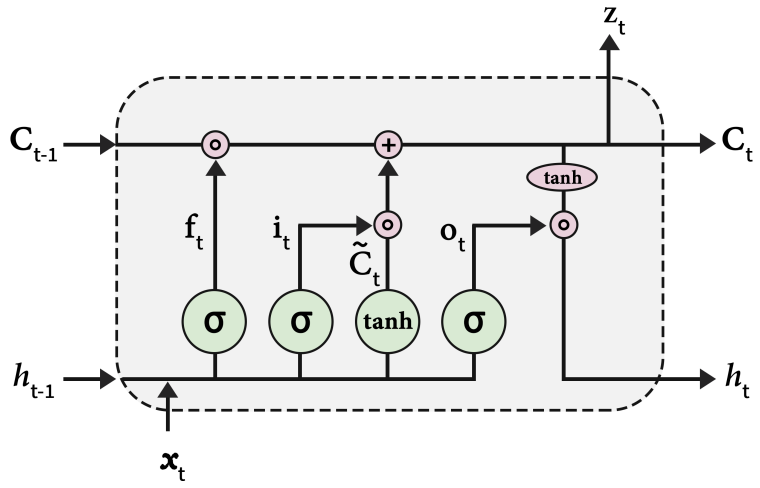


FIGURE 4.6: A long-short term memory unit, diagram adapted from Olah (2015)

4.3.3 Gated Recurrent Units

A Gated Recurrent Unit (GRU) introduced by Chung et al. (2014) is like an LSTM unit but its with forget and input gates combined into a single "update gate" z_t (Goodfellow, Bengio, and Courville, 2016). The gates in LSTMs contain trainable weights requiring large amount of data and computation to train effectively so reducing the number of gates required to address the long-term dependency problem (and therefore reducting the number of trainable parameters in the unit) can improve the results of using RNNs on sequence modeling problems. A GRU is defined as follows:

$$
\begin{aligned}
z_t &= \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \\
r_t &= \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \\
\tilde{h}_t &= \tanh(W_{xr}[r_t \odot h_{t-1}] + W_{hx}x_t]) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t
\end{aligned}
$$

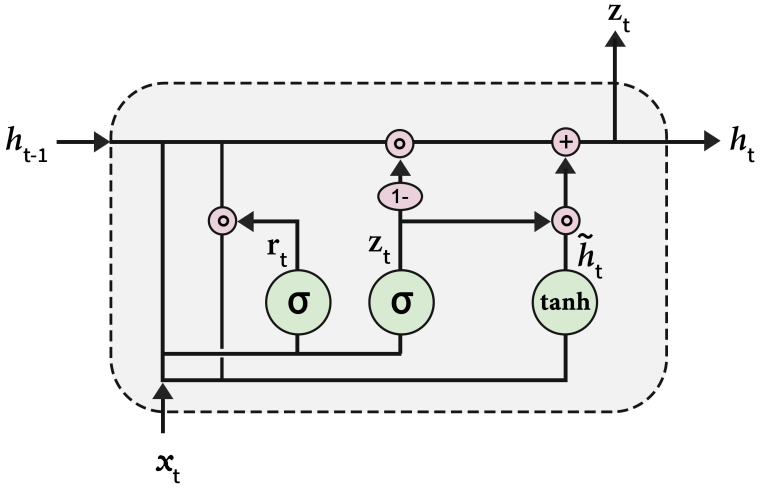


FIGURE 4.7: A gated recurrent unit, diagram adapted from Olah (2015)

4.3.4 Stacked RNNs

Like fully connected layers, RNN layers can be "stacked" on top of each other to create "Deep RNNs" by using the hidden state of one layer as the input to an RNN in the subsequent layer. Deep RNNs were introduced for speech recognition by Graves, Mohamed, and Hinton (2013) and are used to add more representational power and model capacity to sequence modeling tasks. Stacked LSTMs have been used to achieve state-of-the-art in several sequence modeling tasks including sentence dependency parsing (Andor et al., 2016) and video classification, for example in the DeepVideo model by Ng et al. (2015) discussed in the following section.

4.4 Spatial-then-temporal DNNs

Instead of jointly modeling spatio-temporal features using a 3D CNNs, an alternative approach is to decompose the problem and first extract spatial features from frames and then model temporal information using the extracted spatial features. The approaches to video classification that model frames independently using image classification methods together with a majority voting algorithm to combine frame-level classification outputs into a video-level prediction are the simplest case of spatial-then-temporal models. The spatial extraction algorithm compresses frame information into a lower-dimensional feature vector which reduces the amount of motion information that can be captured in the temporal model but the spatial-then-temporal approach can outperform spatio-temporal models (especially with limited data) because they generally require learning fewer parameters and can effectively leverage transfer learning.

The first spatial-then-temporal model to use a DNN used the image processing SIFT algorithm to extract spatial features from frames which were then input into an LSTM to classify actions in soccer videos (Baccouche et al., 2010). Follow-on work by the same authors used a 3D CNN on small 9-frame clips as a spatial feature extractor and an LSTM as a temporal model to classify actions in the KTH dataset containing videos between 8 and 59 seconds of people performing one of six actions. The spatial and temporal models can be combined into a single end-to-end model. When this is done with a CNN spatial model and a RNN temporal model, the combined model is called a "Long-term Recurrent Convolutional Network (Donahue et al., 2014).

4.4.1 Long-term Recurrent Convolutional Networks

Long-term Recurrent Convolutional Networks (LRCNs) were introduced by Donahue et al. (2014) to classify actions in the UCF-101 dataset. They pre-train a variant of VGGNet on the 1.2 million image ILSVRC training subset to initialize their spatial CNN model and then feed their CNN outputs into a stack of 4 LSTM sequence models to produce a variable length prediction that scales to arbitrarily long sequences. Critically, the sequence model weights are re-used at each time step which forces the model to learn generic temporal dynamics which also prevents the parameter count from growing proportionately to the maximum video length. Despite weight re-use as a form of regularization, aggressive dropout of 0.9 was required to avoid overfitting. The LRCN model (shown in Figure 4.8) is trained on 16 frame video clips and produces frame-level predictions at each time-steps which are averaged to produce a clip level prediction that is further averaged across clips to produce a video-level prediction. The achieve state-of-the-art results on UCF-101 activity recognition.

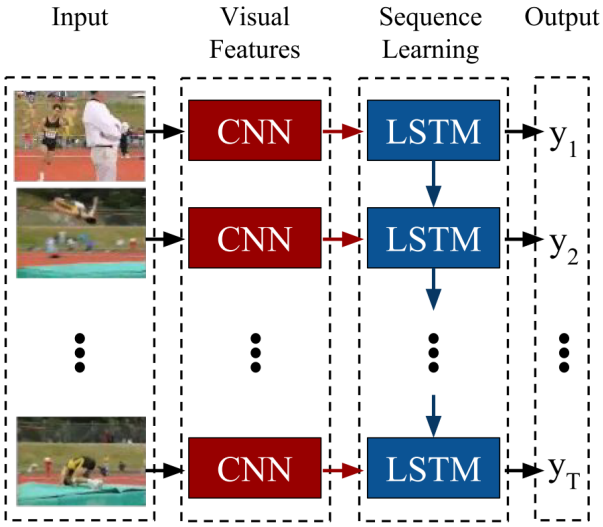


FIGURE 4.8: Architecture of the Long-term Recurrent Convolutional Network (Donahue et al., 2014) for video activity recognition

DeepVideo (Ng et al., 2015) compares various spatial and temporal model combinations for classifying actoins in the UCF-101 and Sports-1M datasets. They compare two ImageNet-winning CNN variants AlexNet and GoogLeNet, both pre-trained on ImageNet, as spatial models and compare the various pooling approaches from Karpathy et al. (2014) with different stacked LSTM models. They find GoogLeNet outperforms AlexNet as a spatial model and find using max-pooling with two fully-connected layers on top of their spatial model outperforms their best LSTM model which was stacked 5 layers deep. They achieve state-of-the-art results on both Sports-1M and UCF-101.

Chapter 5

Data and Implementation

5.1 Data Description

Two video datasets were used in this study. The classification task in the first dataset is to determine the penguin actions during each frame of the video (temporal action localization) and the classification task in the second dataset is to determine the presence of seals in the video frame.

FIGURE 5.1: A photo of a penguin with a camera unit strapped to its back

The first dataset analyzed consists of a relatively small 9 hours of video captured at 60 Frames Per Second from the perspective of penguins swimming in the ocean. This was captured using "animal-borne cameras". Replay XD 720 action cameras were customized with a pressure-proof marine grade tube-shaped casting weighing 100g. The animal-borne camera unit is shown in Figure 5.1. Several thousand hours of video footage was collected but only a small 14 hour subset annotated (manually by a subject expert) and only 10 hours

of footage was usable after removing footage that was too blurry or dark. Our study is similar to that in Mcgregor et al. (2015) which captured 98 hours of footage and manually classified cat actions into one of 7 classes. Here we classify penguin actions into one of 7 classes. The frame class distribution across all videos at 4 FPS is shown in Table 5.1. below. The dataset was split by randomly selecting two full-length videos for the validation set and one for the test set. There are very few datapoints in the breath class. The beginning of ascent looks like end of descent and the entirety of the bottom class. Similarly search and subsurface both look like the breath class at times.

	train	validation	test
ascent	6394	769	733
bottom	4297	269	244
breath	259	32	4
descent	6030	654	524
search	14017	2493	1649
shallow	7557	2317	278
subsurface	3663	666	168

TABLE 5.1: Penguins dataset class distribution across train, validation and test splits

5.1.2 Seals

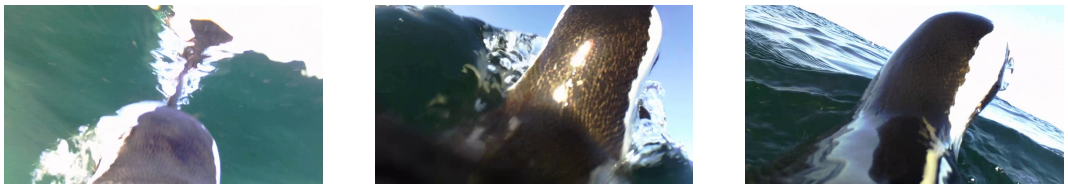
The second dataset is a relatively small 3.5 hours from research that used an underwater video system to study seal behaviour at a salmon trap net fishery in north east Scotland in 2015 aimed at reducing conflict between fisheries and seals. Cameras were placed inside static coastal nets to monitor seals as they moved in and out of nets to depredate salmon. The floating containers that housed the hard-drives and batteries were recovered at the end of each week or sooner if bad weather was forecast, to recover data and recharge batteries. The trial was conducted at a site called Crovie in Gamrie Bay. Research by Reis-filho and Jose (2017) use underwater cameras as nondestructive tool to assess fish assemblages in clearwater amazonian rivers but analyze the footage manually using humans annotating frames second by second. The class distributions and number of frames in each class are shown in Table 5.2 below. The train and validation sets are balanced but the test set happened to have much more seal footage. The data was split by randomly assigning five full-length videos to the validation set and two to the test set:

(A) search

lfig:search

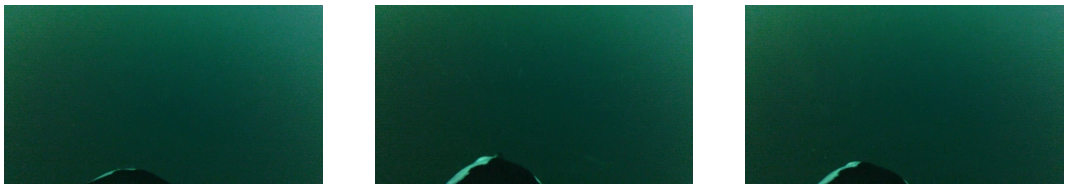
(B) shallow

(C) subsurface



(D) breath

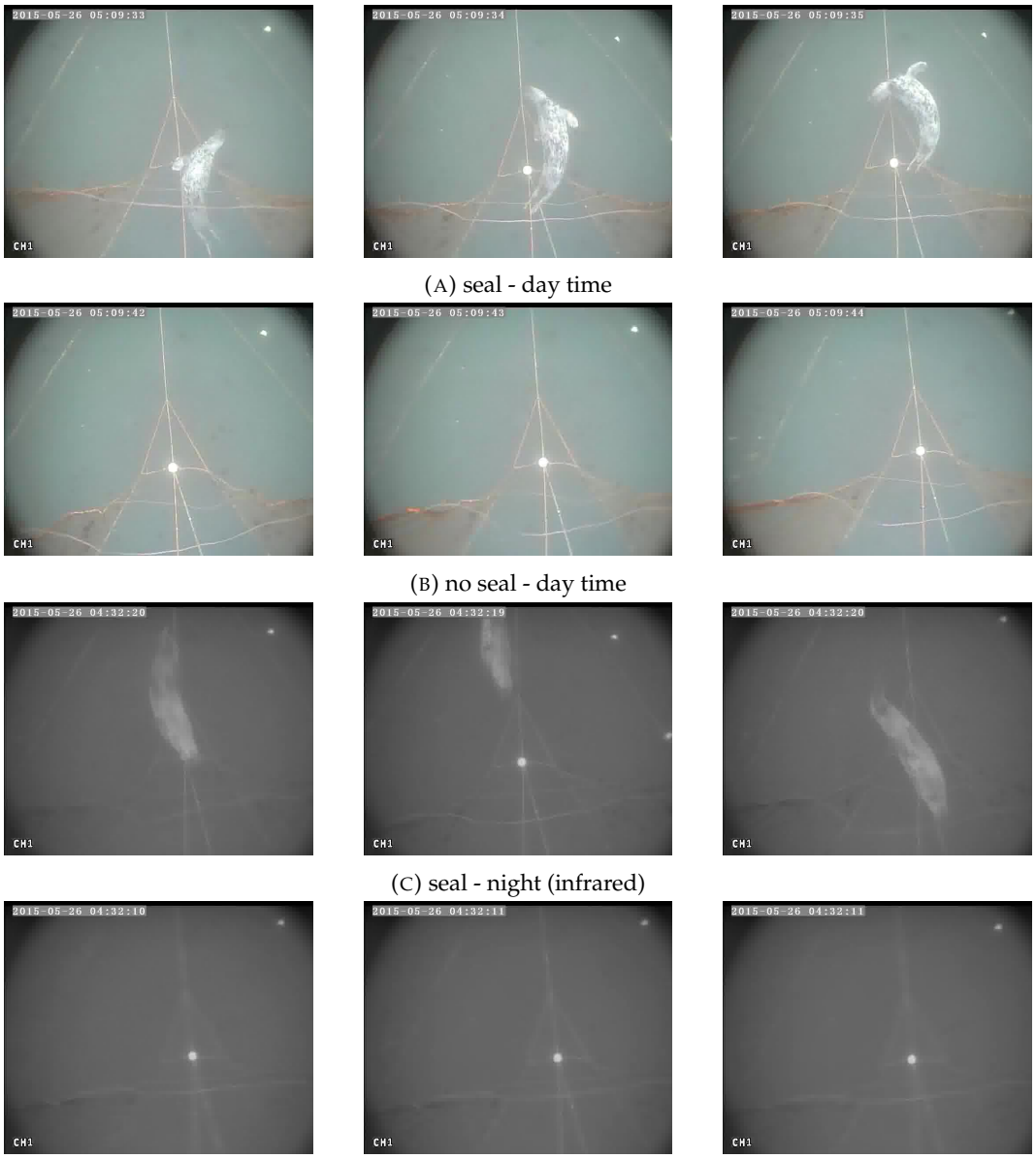
(E) descent



(F) bottom

(G) ascent

FIGURE 5.2: Continuous clips of frames with the same label from the penguins AVED video dataset



(A) seal - day time

(B) no seal - day time

(C) seal - night (infrared)

(D) no seal - night (infrared)

FIGURE 5.3: Continuous clips of frames with the same label from the seals video dataset

	train	validation	test
seal	3826	407	192
no seal	6949	973	111

TABLE 5.2: Seals dataset class distribution across train, validation and test splits

5.2 Model Architectures

The purpose of this study is to evaluate the effectiveness of deep neural networks for classifying all of the frames in a video and compare approaches that utilize the temporal nature of video against approaches that model frames independently. The previous two chapters

presented several architectures for image and video classification that have been successfully used on other datasets and several of the ImageNet-winning CNN architectures will be evaluated on the seals and penguins datasets. The evaluation includes the image classification baseline and video frame concatenation approaches described in Chapter 4. We consider four broad classes of models. One ignores the temporal aspect of video data and treats each frame as a temporally independent image. The second model uses pre-trained CNNs to extract features and concatenates these into a neural network. The third model separates spatial and temporal aspects using a RNN on top of CNN features with various stacking and depth configurations. The final model jointly models spatial and temporal aspects using a 3-dimensional CNN that convolves over space and time.

1. **Image Classification Baseline**
   Classifying frames individually using transfer learning with various pre-trained image classification CNNs and fine-tuning architectures

2. **Concatenating Frame CNN Vectors**
   Classifying frames using concatenated vector encodings of sequences of frames from various pre-trained CNNs and fine-tuning architectures

3. **Long-term Recurrent Neural Networks**
   Classifying frames using video clips by encoding the frames in the clip and passing these to various recurrent model architectures

4. **3D Convolutional Networks**
   Classifying frames using video clips modeled by a 3D convolutional model

The following pretrained CNN models are evaluated: VGG16, ResNet50 and Inception-ResNet v2.
For the image classification baseline and frame concatenation models, multi-layer perceptrons were fit on top of the CNN frame encodings (or concatenated frame encodings) with up to 3 dense layers with varying layer sizes. Dropout was added to varying extents and the ReLU activation function is used together with a softmax cross-entropy error function. An example of the proposed LRCN modification approach with a different pretrained CNN frame encoder (ResnNet50 vs VGG16 used in paper) a different sequence model architecture (stacked 2-layer RNN instead of single layer LSTM) is shown in Figure 5.4 (stacked recurrent model).

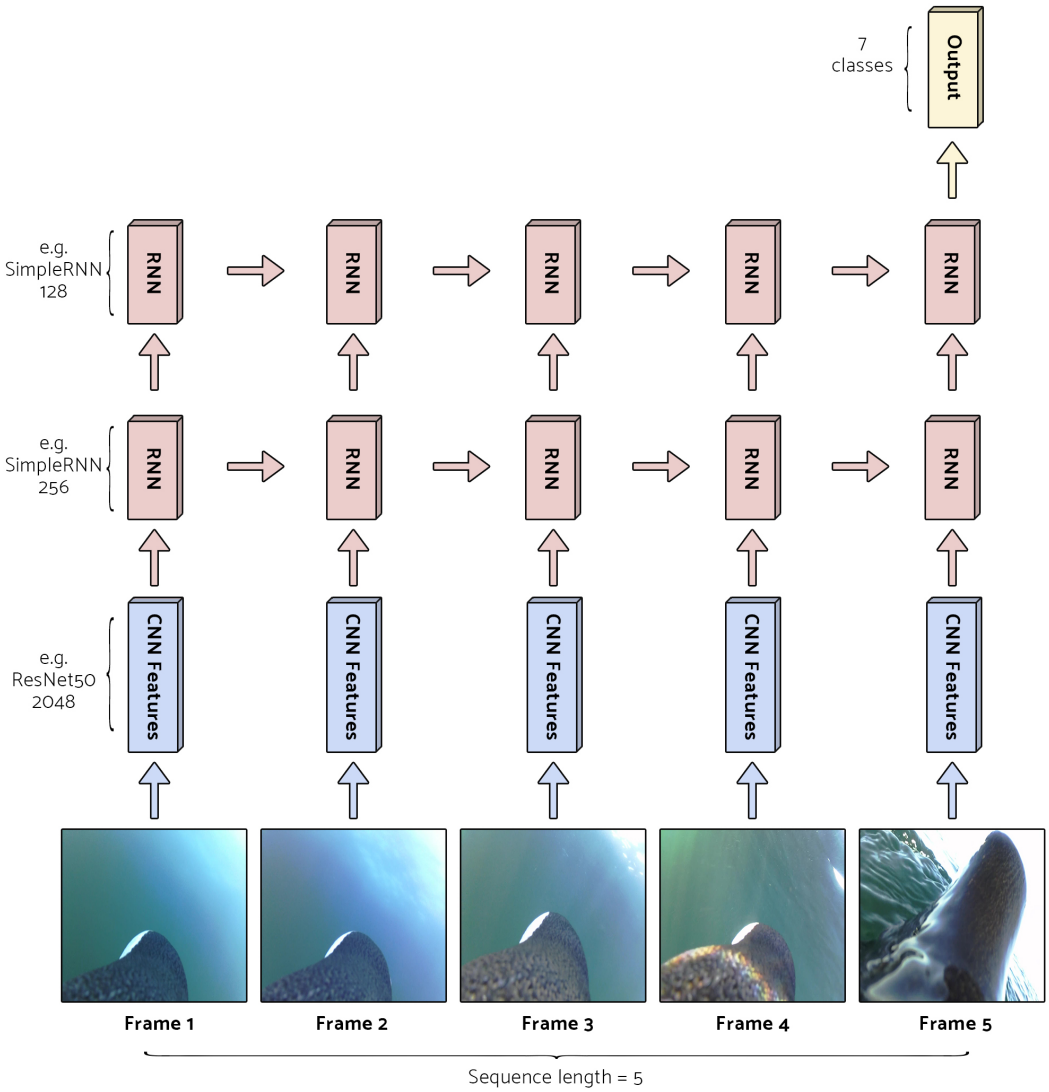


FIGURE 5.4: Modified LRCN Architecture with 2 stacked layers of RNNs applied to penguins dataset with sequence length of 5 using an LSTMs as the RNNs and VGGNet as the CNN

5.3 Implementation

The video was pre-processed by extracting frames at a given frame rate, converting to RGB matrices and pre-computing video sequence blocks of a given sequence length.

The models evaluated in this study were all implemented using the python programming language. The deep neural network models were implemented using the TensorFlow (Abadi et al., 2016) library with Keras (Chollet et al., 2015) which uses automatic differentiation to enable the user to define neural network model as computational graph and have backpropagation computed automatically.

A grid-search shown in Table 5.3 was conducted concurrently on three separate Linux virtual machine instances running on *Google Cloud Platform*, each with eight *Nvidia Tesla K80* Graphics Processing Units (GPUs), 160 GB of RAM and 32 CPU cores. A custom parallel

computing experimentation framework was developed for multi-GPU and multi-instance grid-search utilizing Google Cloud Storage as a shared file-system.

Following Krizhevsky, Sutskever, and Hinton (2012), each model's weights were initialized using the Xavier initialization and each model was trained in 3 rounds of 20 epochs with an early stopping patience of 5 epochs using the Adam optimizer (Kingma and Ba, 2014). The learning rate was initially set to 0.001 and reduced by a factor of 10 between training rounds.

	min	max	step
layer 1 size	32	512	32
layer 2 size	32	512	32
layer 3 size	32	512	32
dropout	0	0.5	0.1
sequence length	1	40	2
pooling	max	average	-
sequence model	LSTM	RNN	GRU
pretrained model	VGG16	ResNet50	InceptionResNetv2

TABLE 5.3: Grid search configurations

Computational Considerations & Optimization

The penguins dataset frame images are approximately 12 GB. The input into the sequence models is a tensor with shape *(batch size, sequence length, frame width, frame height, 3)* and thus is approximately sequence length times larger than the total frame images dataset. The amount of memory on *Google Cloud Platform* instances can be increased and decreased (in contrast to instances on Amazon Web Services which is fixed) and the grid-search encountered out-of-memory errors with 60 GB of memory even when training a single LRCN model with *sequence length* of 5.

Several optimizations were made to improve the memory and time requirements of the grid search. One key optimization was to pre-compute the frame vector encodings from the pre-trained CNN models so that these did not have to be re-computed in each model. A single training epoch for the original LRCN architecture with a VGG encoder takes approximately 15 minutes without pre-computation but only 3 seconds with pre-computed features (because most of the computation time is spent in the CNN part of LRCN). Precomputing the vector encodings for all pre-trained models for all frames of the penguins dataset took approximately 29 hours. After the CNN vectors were precomputed, they were

able to be loaded into memory for each model concurrently.

Even after increasing the memory of the virtual machines to 160 GB, there were still out-of-memory errors when trying to train the C3D model which takes tensors with the shape *(batch size, sequence length, frame width, frame height, 3)* as input and cannot leverage pre-computation. As such, it was not possible to load the full dataset into memory and generators had to be used to stream the data from disk. Constructing sequences dynamically from frames during each generator iteration took several minutes per batch so a memory-mapped tensor is precomputed on disk and the generator streams batches from the memory map. Despite this optimization, a single training epoch for C3D on the penguins dataset took approximately 2 hours and 20 minute and the model took approximately 3 days to converge.

The total processing time on the 24 GPUs was approximately 13 days, implying an approximate total running time of around 300 days on a single GPU system.

Code and analysis scripts are available online at `https://github.com/alxcnwy/Deep-Neural-Networks-for-Video-Classification`.

Chapter 6

Results & Discussion

Deep neural networks using an RNN on top of a CNN frame encoder out-perform other models considered for video classification. A frame-based image classification model was used as a baseline. Long-term Recurrent Convolutional models out-perform other models for both datasets.

We found video models to out-perform image based models and achieved high test accuracy in the penguins dataset achieving 85.81% vs. 80.61% for the image-only model, a 27% relative drop in error. The best performing video model for the seals dataset achieved a 90.17% test accuracy vs. 90.07% for the best image-only model suggesting motion information does not improve seal detection as much. Using a pre-trained CNN as a frame encoder with an RNN on top of a sequence of frames was the best performing model for both cases.

6.1 Results & Discussion

All of the ImageNet-winning CNN models described in Chapter 3 were combined with each RNN model presented in Chapter 4. The best performing model for both Penguin and Seal classification was an RNN combined with a CNN. For the Penguins dataset, an LSTM with 2 layers on top of VGG16 and a sequence length of 20 outperformed all baselines and other CNN/RNN combinations. For Seals, the best performing model was a 2 layer LSTM on top of ResNet50 encoded blocks of 5 frames. In both cases, max-pooled CNN features out-performed average pooled CNN features.

Gomez Villa, Salazar, and Vargas (2017) tried using pre-trained models to classify 26 class Serengeti animal images captured using animal camera traps. ResNet50 features performed best. Siddiqui et al. (2018) also used pre-trained models to classify fish underwater again with ResNet features performing best. Lower frame rates tend to out-perform since they enable the model to observe a greater context from longer input clips. Weinstein (2018)

use Inception models to classify animal vs. background frames finding it to vastly outperform alternative models. This work goes far beyond prior applications of deep neural networks to video classification in ecology applying recurrent models to CNN frame encodings. The best Penguins model has approximately 5'000'000 trainable parameters and the Seals model has approximately 1'400'000 trainable parameters. Donahue et al. (2014) found the most influential hyperparameters to be the number of hidden units in the LSTM RNN models and also which pretrained model was used to extract features. As in our case, they find ResNet and VGG perform best.

Tables with best 5 video models for each dataset, together with confusion matrices, are given below. The best image only model is given for comparison together with the best performing frame concatenation model. Train accuracy is slightly above validation accuracy as expected but there is a relatively high discrepancy between test and validation accuracy across all models for Penguins. Deep neural networks can tend to overfit small datasets and that may be what happened here. We experimented with dropout and other regularization methods to reduce overfitting as much as possible. It is also interesting that VGG16 is the best encoder for the video models but resnet50 is best for the image model, suggesting that a simpler CNN model together with an RNN can perform well while a frame model with no motion context requires a more complex CNN. The best frame concatenation model slightly under-performs the best image only model and the best video model significantly outperforms image-only models.

Accuracy (Valid)	95.01%	94.95%	94.56%	93.40%	93.78%	94.80%	94.06%
Accuracy (Train)	96.22%	95.93%	97.83%	96.89%	95.08%	96.17%	95.42%
Accuracy (Test)	85.81%	83.94%	83.78%	82.54%	81.23%	80.61%	79.96%
Sequence Length	20	20	20	10	10	1	3
Architecture	LRCN	LRCN	LRCN	LRCN	LRCN	IMAGE	C.CAT
Train Time (s)	570	663	1419	693	458	190	242
Pretrained Model	vgg16	vgg16	vgg16	vgg16	vgg16	resnet50	resnet50
Num. Features	512	512	512	512	512	2048	2048
Pooling	max	max	max	max	max	max	max
Sequence Model	SimpleRNN	SimpleRNN	LSTM	LSTM	SimpleRNN	NONE	NONE
Sequence Layers	2	2	2	2	2	NA	NONE
Layer 1 Size	256	256	256	256	256	128	128
Layer 2 Size	128	512	512	512	128	256	128
Layer 3 Size	256	256	256	256	0	256	128
Dropout	0.20	0.20	0.20	0.20	0.20	0.20	0.20
Model Parameters	903 559	3 214 087	4 985 863	3 675143	247 047	362 887	820 487

TABLE 6.1: Top 5 Most Accurate Models on Validation Set for Penguins Data together with the best Image-only model

	ascent	bottom	breath	descent	search	shallow	subsurface
ascent	0.74	0.08	0	0.03	0.01	0.14	0
bottom	0.08	0.43	0	0.46	0	0.04	0
breath	0	0	0	0	1.00	0	0
descent	0.01	0.06	0	0.69	0.01	0.24	0
search	0	0	0	0	0.99	0.01	0
shallow	0.11	0.01	0	0.19	0.03	0.62	0.04
subsurface	0	0	0	0	0.04	0.49	0.48

TABLE 6.2: Normalized confusion matrix for best image-only model on penguins test set

	ascent	bottom	breath	descent	search	shallow	subsurface
ascent	0.79	0.10	0	0.01	0	0.09	0
bottom	0.07	0.57	0	0.36	0	0	0
breath	0	0	0	0	1.00	0	0
descent	0	0.02	0	0.86	0.01	0.11	0
search	0	0	0	0	1.00	0	0
shallow	0	0	0	0.33	0.02	0.63	0.01
subsurface	0	0	0	0.01	0.03	0.48	0.48

TABLE 6.3: Normalized confusion matrix for best spatio-temporal video model on penguins test set

The confusion matrices above show that the image-only model tends to do well at predicting search (which is visually distinct and does not require temporal information to discern) but confuses ascent with bottom or shallow and similarly for ascent, which is understandable given the lack of temporal information. The video model does better on these classes, benefitting from the temporal information in its inputs which is required to differentiate ascent from descent and vice versa.

The accuracy and model parameters for the top 5 video models on the Seals dataset together with the best image and frame concatenation models are shown below. There is a reasonable difference between train, validation and test performance. The best performing model on the test set is a video model but it only marginally outperforms the best image model, suggesting the additional performance in using motion information to detect seals in camera trap footage is marginal.

Accuracy (Valid)	96.18%	96.18%	96.06%	96.03%	96.03%	92.64%	95.10%
Accuracy (Train)	94.11%	93.90%	92.93%	92.33%	93.80%	93.31%	95.66%
Accuracy (Test)	89.83%	90.17%	88.63%	88.81%	87.80%	90.07%	88.96%
Sequence Length	5	5	3	5	5	1	3
Architecture	LRCN	LRCN	LRCN	LRCN	LRCN	IMAGE	C.CAT
Train Time (s)	665	558	974	577	502	264	266
Pretrained Model	resnet50	resnet50	resnet50	resnet50	resnet50	resnet50	resnet50
Num. Features	2048	2048	2048	2048	2048	2048	2048
Pooling	max	max	max	max	max	max	max
Sequence Model	LSTM	LSTM	LSTM	GRU	LSTM	NONE	NONE
Sequence Layers	2	2	2	2	2	NA	NA
Layer 1 Size	128	512	512	512	256	128	128
Layer 2 Size	128	512	512	512	128	512	128
Layer 3 Size	256	256	128	512	0	128	512
Dropout	0.20	0.20	0.20	0.20	0.20	0.20	0.20
Model Parameters	1 410 818	8 000 258	7 541 122	6 820 354	394 887	632 490	870 146

TABLE 6.4: Top 5 Most Accurate Models on Validation Set for Seals Data together with the best Image-only model

	no seal	seal
no seal	0.99	0.01
seal	0.17	0.83

TABLE 6.5: Normalized confusion matrix for best image-only model on seals dataset

	no seal	seal
no seal	0.95	0.05
seal	0.15	0.85

TABLE 6.6: Normalized confusion matrix for best spatio-temporal video model on seals dataset

The confusion matrices for the seals case show the video model does well when no seal is present but mis-classifies seal frames as having no seal, and slightly worse than the image-only model while being better at the no-seal class than the image model.

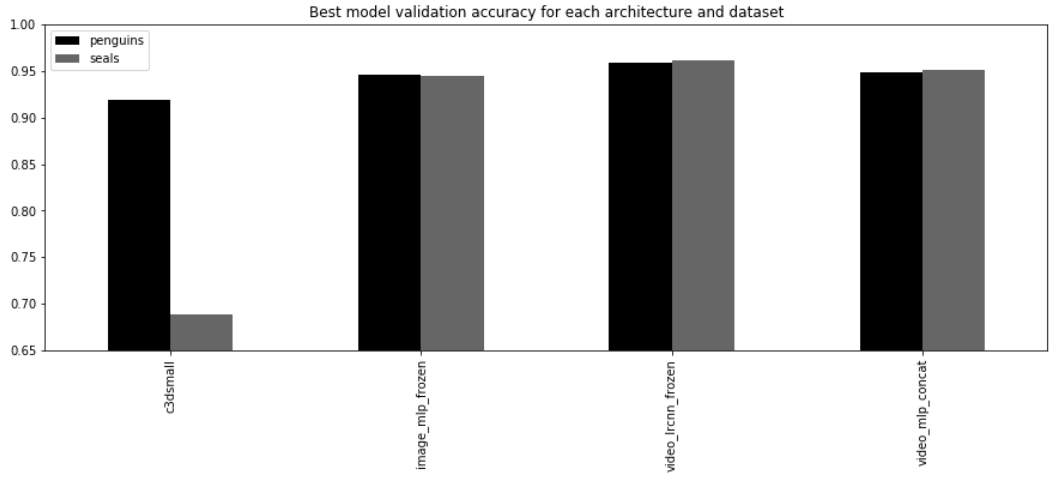


FIGURE 6.1: Best validation accuracy for each architecture in each dataset

The best model for both datasets is the LRCN followed by the image-only, concatenation models and then C3D - presumably because the datasets are too small to fit the much more complex 3D CNN model.

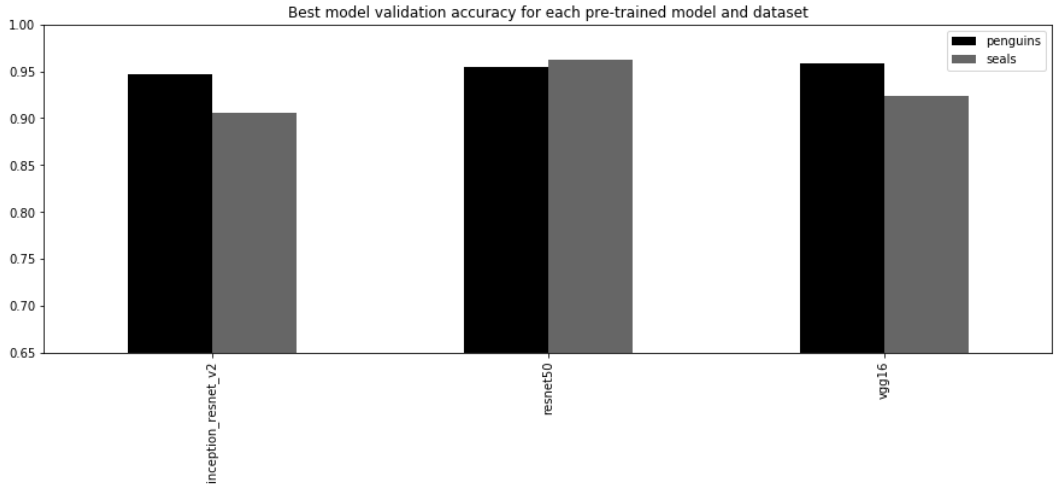


FIGURE 6.2: Best validation accuracy for each pre-trained model in each dataset

ResNet50 is the best performing seals model while VGG16 works best for Penguins and the more complex Inception-Resnet-V2 underperforms both. It is interesting to note that the best Penguins model uses a simpler CNN with a more complex LSTM RNN while the reverse is true for the best Seals model which uses Resnet50 together with a simple RNN. I expect this has to do with the limited dataset size and that a more complex CNN would perform better together with more complex RNN architectures if more data was available to enable the models to better take advantage of temporal information in the input videos.

Long-term Recurrent Convolutional deep neural networks tend to out-perform shallower models. One of the top 5 Penguins models used only 10 frames as input while the rest use 20 frames. The length required depends on the action in the video clip. For penguins, the action is clear within a few seconds or around 10 frames. For Seals a lower length of 3 performs best but for other tasks, a longer sequence length may be required. Generally the sequence length should be as long as is required for a human to identify the class.

Longer input sequences tend to outperform in Penguins but not in Seals classification. Longer sequences tend to contain more context especially when trained with lower frame rates. The Seals data has seals swimming quickly and as such a higher frame rate and lower sequence length work well.

6.1.1 Frame Error Analysis

In order to better understand the kinds of errors the models make, we constructed the plots below - for each of the best video and image-only model - which show the model's predicted class probabilities together with the actual prediction (indicated by line colour) for every timestep in the Seals dataset and the first 1000 timesteps in the Penguins dataset. Crosses indicate incorrect predictions.
The plots show that the errors in the Seals dataset occur in the first few and last few frames of each seal visit. It also shows the video model predictions are less noisy than the image model predictions. Inspecting the videos, it's unclear even to a human at the very start and end of each visit whether a seal is in fact in view (the annotator could see a seal mid-visit then use that information to go back and label the first and last few frames). The result is encouraging though as seal visits are being detected even if only once the seal is more fully in view.

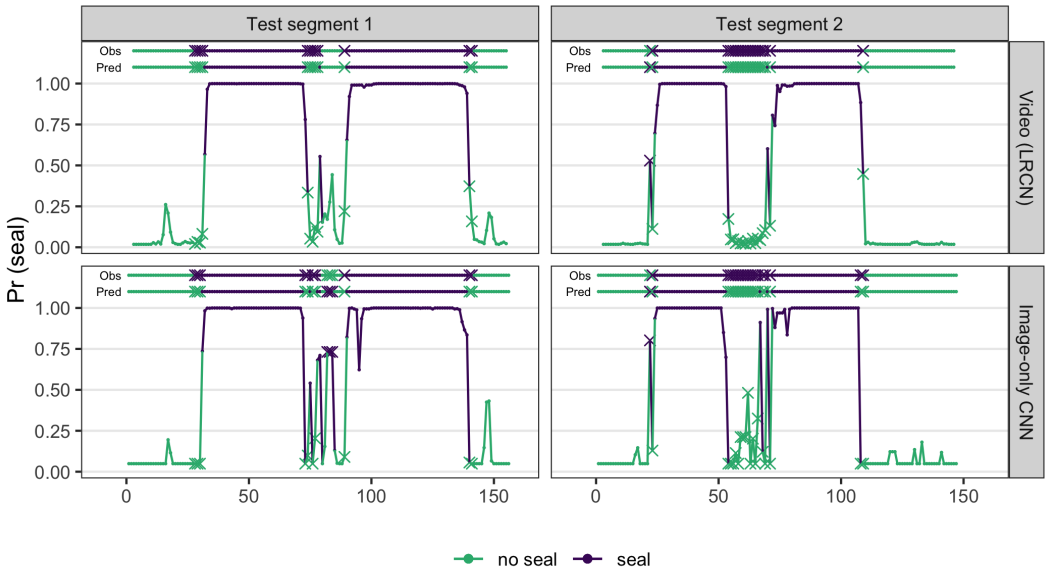


FIGURE 6.3: Analysis of which frames errors occur in the seals test set

The plot below shows again that the video model predictions are somewhat less noisy than the image model predictions. It also shows that errors tend to occur when the model predictions are noisy which is encouraging since it would be easy to flag these noisy regions for a human to review the dataset at these points and correct the model's output, suggesting that even if the model output isn't perfect it could still be very useful.

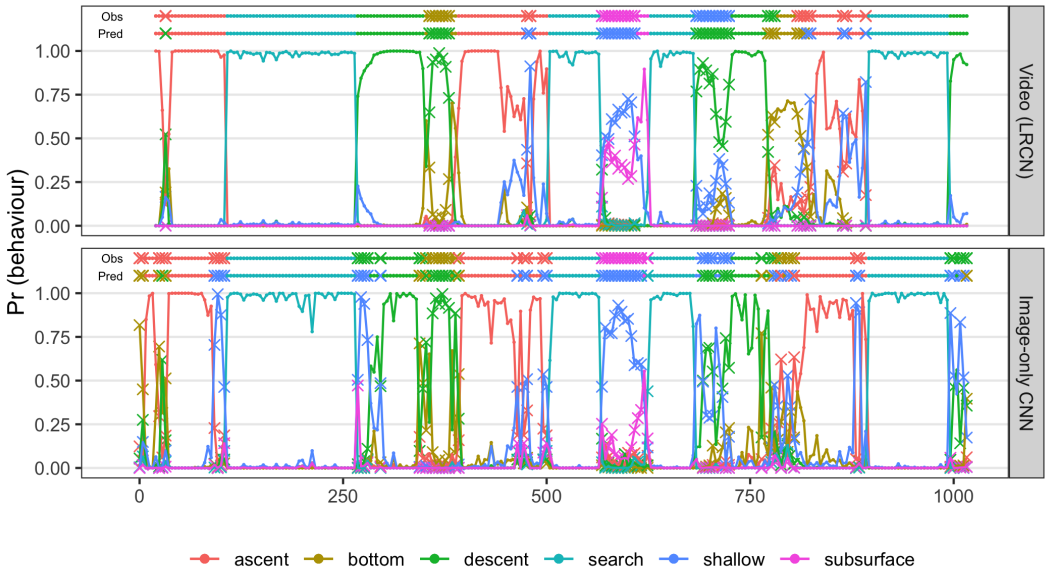


FIGURE 6.4: Analysis of which frames errors occur in the penguins test set

Chapter 7

Conclusion

Video classification is a challenging and important task in ecology, made difficult by the high dimensionality of video data and limited labeled dataset sizes. Although images are more commonly used in ecological research and are easier to work with (Swinnen et al., 2014), movement information contained in video enables better differentiation between animal behaviours and taking movement into account significantly improves the identification of animals and their behaviours (Trinh et al., 2016). Including temporal information in video data significantly reduced classification error compared to an image-only model for both the penguins dataset but was only marginally better for the seals dataset, suggesting video models are more powerful for complex video classification tasks like penguin actions than the simpler seal detection task. The datasets analyzed were relatively small, consisting of only 6-12 hours of labelled footage each, and the ability of the models to generalize to new environments is unclear. The relative out-performance of video models will likely be larger with larger datasets.

Video data is manually annotated by recording the start and end times of events of interest to create a one-to-one mapping between frames and outcome classes, with each frame allocated to a single class based on its timestamp. The primary difficulty is that most video data is recorded at frame rates far exceeding the timescales at which animal behaviour changes. There is substantial subjectivity in determining when one behaviour stops and another starts, even when the events are in principle clearly separable, as in the case of seal visits to salmon nets, or distinguishing descending from bottom dives. The boundaries between more complex behavioural classes, such as commuting and foraging, may be even more difficult to discern. Video models are less sensitive to class ambiguity than image-only models, because neighbouring images provide additional context, but poorly separated classes and ambiguous annotations cause problems for any classifer and place an effective upper limit on the accuracy that can be achieved. Video data should be annotated with broader research objectives in mind, wherever this is possible to do. For seal visits, for example, the detection of a seal presence is more important that identifying the exact time of entry. The first and last few frames of a visit often contain only a tiny sliver of seal

or, because the times are approximate, no seal at all.

Deep neural networks are a powerful family of models for classifying video. We found that pre-trained CNN models used in combination with RNN models out-perform image based models for the datasets we analyzed. When building an RCNN, key choices are what frame rate and sequence length to use. These factors are study-specific, and the chosen frame rate need not be the same as the frame rate used to convert video to frames. Video classification networks use frame $t, t-1, \ldots, t-F$ to predict the class of frame t, so a higher frame rates allow for fine-scale changes in movement to be captured, but F frames covers a shorter time interval. Increasing sequence length requires more parameters, increasing the chances of overfitting and requiring more data. Which of the two – looking back further in time or capturing fine-scale movement – improves classification accuracy more will be study-specific. These factors can be investigated by searching over possible frame rate/length pairs, but this quickly becomes computationally expensive. Our applications have relatively little labelled data and so we fixed the frame rate to one that would allow broad differences in behaviour, observed over a few seconds, with $5 < F < 10$. Pre-trained CNNs offer a parsimonious way of summarizing images into compact vector representations that can be passed on the second-stage RNN to capture motion information (Donahue et al., 2014). Since model complexity is primarily achieved through more parameters, this balance reflects goal of reducing validation error through model parsimony.

We are optimistic about the potential for deep neural networks to solve complex video classification tasks and hope that the software developed and open-sourced during this research will accelerate the adoption of DNNs for video classification in ecology.

7.1 Directions for Future Work

An ensemble model was not trained but we believe it will perform better at higher computational cost should sufficient data be available. Rather than predicting each frame in isolation smoothing frame predictions based on majority votes of the neighbouring frames was left as future work.

Data Augmentation

Both the Seals and Penguin datasets could be artificially increased by augmenting the data with horizontal flips, rotations and randomized colour changes.

Bidirectional Sequence Models

Only one directional RNNs were considered but RNNs stacked with recurrence in both temporal dimensions could improve performance. This would not be possible to run in practice if real-time performance is required but could improve accuracy.

Better hyper-parameter search

Neural architecture search (Zoph and Le, 2017) uses reinforcement learning to search for the best performing architecture and could be used to improve results in this work. Genetic algorithms are another potential approach to try.

MobileNet and variants

There is a trade-off between model size and test-time inference speed. Model distillation can be used to learn compressed representations of models or a pre-trained model designed for low computational resources like MobileNet could be used. Predictions could be computed on an edge device and streamed to a central server but latency could be an issue, especially without model compression.

Transfer Learning in Ecology

Like ImageNet models which are trained on massive varied datasets, it might be possible to mix ecological images with the ImageNet dataset to produce a model that outperforms ImageNet pre-training.

Object Detection Models

Using an object-detection model such as Regional-CNN or YOLOv3 could be used to assist the model in where to focus its attention but these require vastly more labelled data.

Multi-agent Models

Multi-agent models such as swarm models can potentially be used in combination with the video classification methods presented in this work in order to model complex systems such as bird migration patterns.

Incorporating Audio and Other Sensor Data

Incorporating audio and other biotelemetry sensor data is kept as future work but presents promising research avenues (Moll et al., 2007).

Appendix A

Convolutional Neural Network Architectures

This appendix contains network architecture diagrams for several CNNs discussed in Chapter 3.

A.1 ResNet

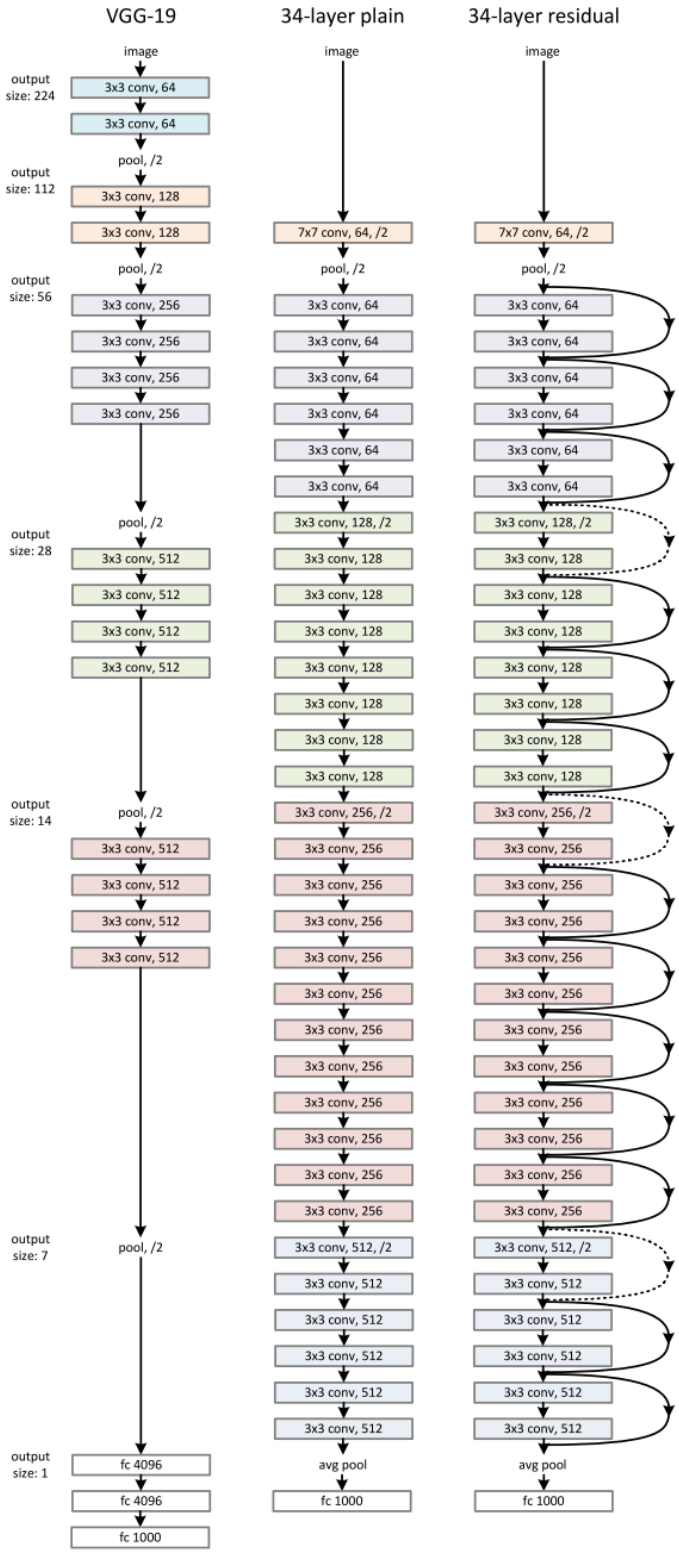


FIGURE A.1: Architecture of ResNet in (He et al., 2015), a Convolutional Neural Network that won the ILSVRC competition in 2015. VGGNet is shown for comparison

A.2 GoogLeNet / Inception v1

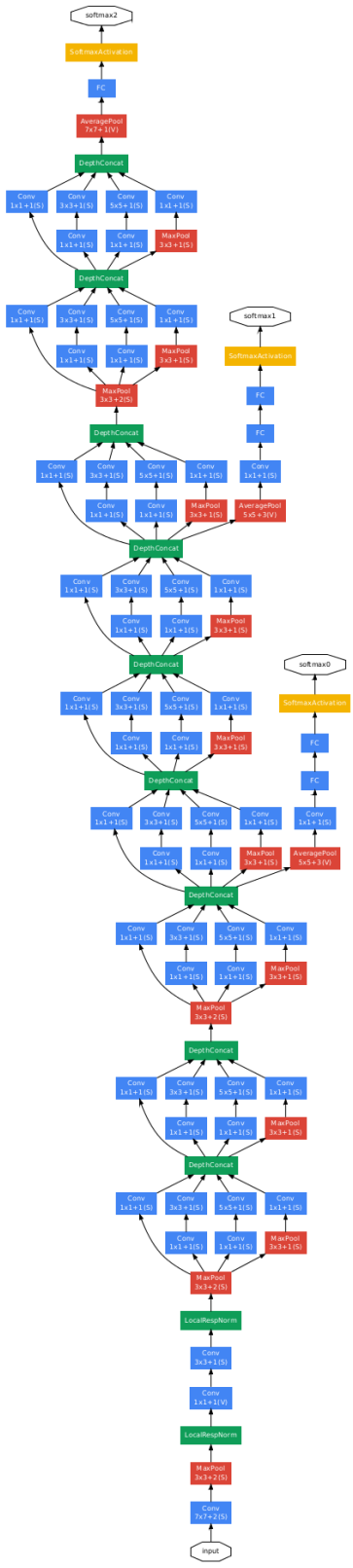


FIGURE A.2: Architecture of GoogLeNet (Szegedy, Liu, et al., 2015), a Convolutional Neural Network that won the ILSVRC competition in 2014.

Bibliography

Abadi, Martin, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. "Tensorflow: a system for large-scale machine learning." In *OSDI*, 16:265–283.

Anderson, T Michael, Staci White, Bryant Davis, Rob Erhardt, Meredith Palmer, Alexandra Swanson, Margaret Kosmala, et al. 2016. "The spatial distribution of African savannah herbivores : species associations and habitat occupancy in a landscape context." *Phil. Trans. R. Soc. B* 371 (1703).

Andor, Daniel, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. "Globally normalized transition-based neural networks." *arXiv preprint arXiv:1603.06042.*

Baccouche, Moez, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt. 2010. "Action classification in soccer videos with long short-term memory recurrent neural networks." *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6353 LNCS (PART 2): 154–159. ISSN: 03029743. doi:10.1007/978-3-642-15822-3_20.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2014. "Neural machine translation by jointly learning to align and translate." *arXiv preprint arXiv:1409.0473.*

Bengio, Yoshua, and Yann Lecun. 1995. "Convolutional Networks for Images , Speech , and Time-Series." *The handbook of brain theory and neural networks* 3361 (10).

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. 1994. "Learning long-term dependencies with gradient descent is difficult." *IEEE transactions on neural networks* 5 (2): 157–166.

Bishop, Chris. 2006. *Pattern Recognition and Machine Learning.* Springer.

Bowley, Connor, Alicia Andes, Susan Ellis-Felege, and Travis Desell. 2017. "Detecting wildlife in uncontrolled outdoor video using convolutional neural networks." *Proceedings of the 2016 IEEE 12th International Conference on e-Science, e-Science 2016:* 251–259. ISSN: 2325-372X. doi:10.1109/eScience.2016.7870906.

Burghardt, Tilo, and Calic Janko. 2006. "Analysing Animal Behaviour in Wildlife Videos Using Face Detection and Tracking." *IEEE Proceedings-Vision, Image and Signal Processing* 153 (3): 305–312.

Chen, Guobin, Tony X. Han, Zhihai He, Roland Kays, and Tavis Forrester. 2014. "Deep convolutional neural network based species recognition for wild animal monitoring." *2014 IEEE International Conference on Image Processing, ICIP 2014*, no. January: 858–862. ISSN: 01675273. doi:10.1109/ICIP.2014.7025172.

Chollet, F. 2017. *Deep Learning With Python.* Manning Publications Co.

Chollet, Francois, et al. 2015. *Keras.*

Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling." *arXiv preprint* arXiv:1412:1–9. http://arxiv.org/abs/1412.3555.

Cortes, Corinna, and Vladimir Vapnik. 1995. "Support-vector networks." *Machine learning* 20 (3): 273–297.

Cybenko, George. 1989. "Approximation by superpositions of a sigmoidal function." *Mathematics of control, signals and systems* 2 (4): 303–314.

Dauphin, Yann N, Razvan Pascanu, Caglar Gulcehre, and Kyunghyun Cho. 2014. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization." *Advances in neural information processing systems:* 1–9.

Davis, R W, L A Fuiman, T M Williams, S O Collier, W P Hagey, S B Kanatous, S Kohin, and M Horning. 1999. "Hunting Behavior of a Marine Mammal Beneath the Antarctic Fast Ice." *Science* 283 (February): 993–996.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. "Imagenet: A large-scale hierarchical image database." In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 248–255. Ieee.

Deng, Li, Geoffrey Hinton, and Brian Kingsbury. 2013. "Deng, L., Hinton, G., & Kingsbury, B. (2013, May). New types of deep neural network learning for speech recognition and related applications: An overview." *Acoustics, Speech and Signal Processing (ICASSP):* 8599–8603.

Dollár, Piotr, Vincent Rabaud, Garrison Cottrell, and Serge Belongie. 2005. "Behavior recognition via sparse spatio-temporal features." VS-PETS Beijing, China.

Donahue, Jeff, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, and Trevor Darrell. 2014. "Long-term Recurrent Convolutional Networks for Visual Recognition and Description." *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition:* 2625–2634. arXiv: arXiv:1411.4389v4.

Duchi, John, Elad Hazan, and Yoram Singer. 2011. "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of Machine Learning Research* 12 (Jul): 2121–2159.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2001. *The elements of statistical learning.* Vol. 1. 10. Springer series in statistics New York, NY, USA:

Fukushima, Kunihiko. 1980. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position." *Biological Cybernetics* 36 (4): 193–202. ISSN: 03401200. doi:10.1007/BF00344251.

Glorot, Xavier, and Yoshua Bengio. 2010. "Understanding the difficulty of training deep feedforward neural networks Xavier." *Proceedings of the thirteenth international conference on artificial intelligence and statistics:* 249–256. ISSN: 1938-7228. doi:10.1021/ct2009208. arXiv: 1701.05369. http://arxiv.org/abs/1701.05369.

Goldberg, Yoav. 2017. *Neural Network Methods for Natural Language Processing.* ISBN: 9781627052986. doi:10.2200/S00762ED1V01Y201703HLT037.

Gomez Villa, Alexander, Augusto Salazar, and Francisco Vargas. 2017. "Towards automatic wild animal monitoring: Identification of animal species in camera-trap images using very deep convolutional neural networks." *Ecological Informatics* 41:24–32. ISSN: 15749541. doi:10.1016/j.ecoinf.2017.07.004. arXiv: 1603.06169.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning.* ISBN: 9780521835688. doi:10.1038/nmeth.3707. arXiv: arXiv:1312.6184v5. http://www.nature.com/doifinder/10.1038/nature14539%7B%5C%%7D5Cnhttp://www.nature.com/doifinder/10.1038/nmeth.3707.

Graves, Alex, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. *Speech recognition with deep recurrent neural networks.*

Gray, Patrick C., Kevin C. Bierlich, Sydney A. Mantell, Ari S. Friedlaender, Jeremy A. Goldbogen, and David W. Johnston. 2019. "Drones and convolutional neural networks facilitate automated and accurate cetacean species identification and photogrammetry." *Methods in Ecology and Evolution* 10 (9): 1490–1500. ISSN: 2041210X. doi:10.1111/2041-210X.13246.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. "Deep Residual Learning for Image Recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition:* 770–778.

Heithaus, Michael R, Greg J Marshall, Birgit M Buhleier, and Lawrence M Dill. 2001. "Employing Crittercam to study habitat use and behavior of large sharks." 209:307–310.

Hinton, Geoffrey, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2012. "Improving neural networks by preventing co-adaptation of feature detectors." *arXiv preprint* arXiv:1207. ISSN: 09702067.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. "Long Short-Term Memory." *Neural Computation* 9 (8): 1735–1780.

Horn, Berthold KP, and Brian G Schunck. 1981. "Determining optical flow." *Artificial intelligence* 17 (1-3): 185–203.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. 1989. "Multilayer feedforward networks are universal approximators." *Neural networks* 2 (5): 359–366.

Howard, Jeremy, and Sebastian Ruder. 2018. "Universal language model fine-tuning for text classification." In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1:328–339.

Hu, Jie, Li Shen, and Gang Sun. 2018. "Squeeze-and-excitation networks." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7132–7141.

Hubel, D, and T Wiesel. 1962. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex." *The Journal of Physiology* 160 (1): 106–154. ISSN: 14697793. doi:10.1113/jphysiol.1962.sp006837.

Hubel and Wiesel. 1968. "Receptive fields and functional architecture of monkey striate cortex." *The Journal of physiology* 195 (1): 215–243.

Ioffe, Sergey, and Christian Szegedy. 2015. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." *arXiv preprint* arXiv:1502. ISSN: 0717-6163. doi:10.1007/s13398-014-0173-7.2. arXiv: 1502.03167. http://arxiv.org/abs/1502.03167.

Ji, Shuiwang, Wei Xu, Ming Yang, and Kai Yu. 2010. "3D Convolutional neural networks for human action recognition." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35 (1): 221–231. ISSN: 01628828. doi:10.1109/TPAMI.2012.59. arXiv: 1102.0183.

Jiang, Yu-Gang, Zuxuan Wu, Jun Wang, Xiangyang Xue, and Shih-Fu Chang. 2018. "Exploiting feature and class relationships in video categorization with regularized deep neural networks." *IEEE transactions on pattern analysis and machine intelligence* 40 (2): 352–364.

Karpathy, Andrej. 2014. "What I learned from competing against a ConvNet on ImageNet." *Disponıvel em http://karpathy. github. io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet. Acessado em* 2.

Karpathy, Andrej, Thomas Leung, George Toderici, Rahul Suthankar, Sanketh Shetty, and Li Fei-fei. 2014. "Large-scale Video Classification with Convolutional Neural Networks." *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition.* 1725–1732. ISSN: 978-1-4799-5118-5. doi:10.1109/CVPR.2014.223. arXiv: 1412.0767.

Keller, James M, Michael R Gray, and James A Givens. 1985. "A fuzzy k-nearest neighbor algorithm." *IEEE transactions on systems, man, and cybernetics,* no. 4: 580–585.

Kelly, Marcella J. 2001. "Computer-aided photograph matching in studies using individual identification: an example from Serengeti cheetahs." *Journal of Mammalogy* 82 (2): 440–449.

Kim, Yoon. 2014. "Convolutional Neural Networks for Sentence Classification." *arXiv preprint* arXiv:1408:1746–1751. ISSN: 10709908. doi:10.3115/v1/D14-1181. arXiv: 1408.5882. http://arxiv.org/abs/1408.5882.

Kingma, Diederik P, and Jimmy Ba. 2014. "Adam: A method for stochastic optimization." *arXiv preprint* arXiv:1412. arXiv: arXiv:1412.6980v8.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. 2012. "ImageNet Classification with Deep Convolutional Neural Networks." *Advances in neural information processing systems:* 1097–1105. ISSN: 10495258. doi:http://dx.doi.org/10.1016/j.protcy.2014.09.007. arXiv: 1102.0183.

Lahiri, Mayank, Chayant Tantipathananandh, Rosemary Warungu, Daniel I Rubenstein, and Tanya Y Berger-Wolf. 2011. "Biometric animal databases from field photographs: identification of individual zebra in the wild." In *Proceedings of the 1st ACM international conference on multimedia retrieval*, 6. ACM.

Laptev, Ivan, Marcin Marszałek, Cordelia Schmid, and Benjamin Rozenfeld. 2008. "Learning realistic human actions from movies."

LeCun, Yann, et al. 1989a. "Generalization and network design strategies." *Connectionism in perspective:* 143–155.

LeCun, Yann, Yoshua Bengio, et al. 1995. "Convolutional networks for images, speech, and time series." *The handbook of brain theory and neural networks* 3361 (10): 1995.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015. "Deep learning." *nature* 521 (7553): 436.

LeCun, Yann, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. 1989b. "Backpropagation Applied to Handwritten Zip Code Recognition." *Neural Computation* 1 (4): 541–551.

LeCun, Yann, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence D Jackel. 1990. "Handwritten Digit Recognition with a Back-Propagation Network." *Advances in neural information processing systems:* 396–404.

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 86 (11): 2278–2324.

LeCun, Yann, Fu Jie Huang, and Léon Bottou. 2004. "Learning methods for generic object recognition with invariance to pose and lighting." *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2. ISSN: 10636919.

Leshno, Moshe, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. 1993. "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function." *Neural networks* 6 (6): 861–867.

Lienhart, Rainer, and Jochen Maydt. 2002. "An Extended Set of Haar-like Features for Rapid Object Detection." *Proceedings. international conference on image processing.* ISSN: 03702693. doi:10.1016/0370-2693(92)90806-F.

Liu, Weibo, Zidong Wang, Xiaohui Liu, Yurong Liu, and Fuad E Alsaadi. 2016. "A survey of deep neural network architectures and their applications." *Neurocomputing* 234:11–26. ISSN: 0925-2312. doi:10.1016/j.neucom.2016.12.038. http://dx.doi.org/10.1016/j.neucom.2016.12.038.

Lowe, David. 1999. "Object recognition from local scale-invariant features." *The proceedings of the seventh IEEE international conference* 2:1150–1157.

Marshall, Greg J. 1998. "Crittercam: an animal-borne imaging and data logging system." *Marine Technology Society. Marine Technology Society Journal* 32 (1): 11.

Mcgregor, Hugh, Sarah Legge, Menna E Jones, and Christopher N Johnson. 2015. "Feral Cats Are Better Killers in Open Habitats , Revealed by Animal-Borne Video." *PloS one:* 1–12. doi:10.5061/dryad.47jp0.

Milletari, Fausto, Nassir Navab, and Seyed-ahmad Ahmadi. 2016. "V-Net : Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation." *Fourth International Conference on 3D Vision:* 565–571. arXiv: arXiv:1606.04797v1.

Moll, Remington J., Joshua J. Millspaugh, Jeff Beringer, Joel Sartwell, and Zhihai He. 2007. "A new 'view' of ecology and conservation through animal-borne video systems." *Trends in Ecology and Evolution* 22 (12): 660–668.

Nair, Vinod, and Geoffrey E Hinton. 2010. "Rectified linear units improve restricted boltzmann machines." In *Proceedings of the 27th international conference on machine learning (ICML-10)*, 807–814.

Ng, Joe, Matthew Hausknecht, Sudheendra Vijaynarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. 2015. "Beyond Short Snippets: Deep Networks for Video Classification." *Proceedings of the IEEE conference on computer vision and pattern recognition.* ISSN: 0094243X. doi:10.1063/1.4791421. arXiv: 1503.08909.

Nielsen, Michael A. 2015. *Neural networks and deep learning.* Vol. 25. Determination press USA.

Olah, Christopher. 2015. "Understanding lstm networks." *colah's blog.*

Ponganis, P J, R P Van Dam, G Marshall, T Knower, and D H Levenson. 2000. "Sub-ice foraging behavior of emperor penguins." *The Journal of experimental biology* 203 (Pt 21): 3275–8. http://www.ncbi.nlm.nih.gov/pubmed/11023847.

Powell, V. 2017. "Image Kernels - Explained Visually Image Kernels - Explained Visually." Accessed January 11, 2019. http://setosa.io/ev/image-kernels/.

Reis-filho, Amorim, and Kurt Schmid Jose. 2017. "Baited remote underwater video as a promising nondestructive tool to assess fish assemblages in clearwater Amazonian rivers : testing the effect of bait and habitat type." *Hydrobiologia* 784 (1): 93–109. doi:10.1007/s10750-016-2860-1.

Ridge, Justin T., Patrick C. Gray, Anna E. Windle, and David W. Johnston. 2020. "Deep learning for coastal resource conservation: automating detection of shellfish reefs." *Remote Sensing in Ecology and Conservation:* 1–10. doi:10.1002/rse2.134.

Rosenblatt, Frank. 1962. "Perceptrons and the Theory of Brain Mechanics." *Cornell AERONAUTICAL LAB INC BUFFALO NY.* VG-1196-G:621.

Rumelhart, David, Geoffrey Hinton, and Ronald Williams. 1986. "Learning representations by back-propagating errors." *Nature* 323 (6088): 533.

Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015a. "Imagenet large scale visual recognition challenge." *International journal of computer vision* 115 (3): 211–252.

Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015b. "ImageNet Large Scale Visual Recognition Challenge." *International Journal of Computer Vision* 115 (3): 211–252. ISSN: 15731405. doi:10.1007/s11263-015-0816-y. arXiv: arXiv:1409.0575v3.

Rutz, Christian, Alex A S Weir, and Alex Kacelnik. 2007. "Video cameras on wild birds." *Science* 318 (5851).

Sanchez, Jorge, Florent Perronnin, and Zeynep Akata. 2011. "Fisher vectors for fine-grained visual categorization." In *FGVC Workshop in IEEE Computer Vision and Pattern Recognition (CVPR).*

Scherer, Dominik, M Andreas, and Sven Behnke. 2010. "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition." *Artificial Neural Networks*, no. September.

Shou, Zheng, Dongang Wang, and Shih-Fu Chang. 2016. "Temporal action localization in untrimmed videos via multi-stage cnns." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1049–1058.

Siddiqui, Shoaib Ahmed, Ahmad Salman, Muhammad Imran Malik, Faisal Shafait, Ajmal Mian, Mark R Shortis, and Euan S Harvey. 2018. "Automatic fish species classification in underwater videos : exploiting pre-trained deep neural network models to compensate for limited labelled data." *ICES Journal of Marine Science* 75:374–389. doi:10.1093/icesjms/fsx109.

Simonyan, Karen, and Andrew Zisserman. 2014. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint* arXiv:1409.

Socher, Richard. 2014. "Recursive Deep Learning for Natural Language Processing and Computer Vision." PhD diss., Stanford.

Soomro, Khurram, Amir Roshan Zamir, and Mubarak Shah. 2012. "UCF101: A dataset of 101 human actions classes from videos in the wild." *arXiv preprint arXiv:1212.0402*.

Swinnen, Kristijn R R, Jonas Reijniers, Matteo Breno, and Herwig Leirs. 2014. "A Novel Method to Reduce Time Investment When Processing Videos from Camera Trap Studies." *PloS one* 9 (6). doi:10.1371/journal.pone.0098881.

Szegedy, Christian, Sergey Ioffe, Vincent Vanhoucke, and Alexander Alemi. 2017. "Inception-v4, inception-resnet and the impact of residual connections on learning." *AAAI* 4:12.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. "Going Deeper with Convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition*: 1–9. ISSN: 10974660. doi:10.1002/jctb.4820. arXiv: 1409.4842.

Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2015. "Rethinking the Inception Architecture for Computer Vision." *Proceedings of the IEEE conference on computer vision and pattern recognition*: 2818–2826. ISSN: 08866236. doi:10.1109/CVPR.2016.308. arXiv: 1512.00567. http://arxiv.org/abs/1512.00567.

Tieleman, Tijmen, and Geoffrey Hinton. 2012. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." *COURSERA: Neural networks for machine learning* 4 (2): 26–31.

Tran, Du, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. 2015. "Learning Spatiotemporal Features with 3D Convolutional Networks." *Proceedings of the IEEE international conference on computer vision:* 4489–4497. arXiv: arXiv:1412.0767v4.

Trinh, Tuan Tu, Ryota Yoshihashi, Rei Kawakami, Makoto Iida, and Takeshi Naemura. 2016. "Bird detection near wind turbines from high-resolution video using lstm networks." *World Wind Energy Conference.*

Turk, Matthew, and Alex Pentland. 1991. "Face Recognition using eigenfaces and SVMs." *Cin.Ufpe.Br:* 586–591. http://www.cin.ufpe.br/%7B~%7Drps/Artigos/Face%20Recognition%20Using%20Eigenfaces.pdf.

Walther, Dirk, Duane R Edgington, and Christof Koch. 2004. "Detection and tracking of objects in underwater video." *roceedings of the 2004 IEEE Computer Society Conference* 1.

Weinstein, Ben G. 2018. "Scene-specific convolutional neural networks for video-based biodiversity detection." *Methods in Ecology and Evolution* 9 (6): 1435–1441. doi:10.1111/2041-210X.13011.

Wender, J. 2015. "Meet Grandfather Flash, the Pioneer of Wildlife Photography." Accessed January 27, 2019. https://www.nationalgeographic.com/photography/proof/2015/11/20/meet-grandfather-flash-the-pioneer-of-wildlife-photography/.

Yang, Ming, Shuiwang Ji, Wei Xu, Jinjun Wang, Fengjun Lv, Kai Yu, Yihong Gong, Mert Dikmen, Dennis J Lin, and Thomas S Huang. 2009. "Detecting Human Actions in Surveillance Videos." *2009 TREC Video Retrieval Evaluation Notebook Papers.* ISSN: 02893770. doi:10.1016/S0010-2180(98)00090-X.

Zeiler, Matthew D., and Rob Fergus. 2013. "Visualizing and understanding convolutional networks." *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8689 LNCS (PART 1): 818–833. ISSN: 16113349. doi:10.1007/978-3-319-10590-1_53. arXiv: 1311.2901.

Zhang, Zhi, Zhihai He, Guitao Cao, and Wenming Cao. 2016. "Animal Detection from Highly Cluttered Natural Scenes Using Spatiotemporal Object Region Proposals and Patch Verification." *IEEE Transactions on Multimedia* 18 (10): 2079–2092. ISSN: 15209210. doi:10.1109/TMM.2016.2594138.

Zoph, Barret, and Quoc V Le. 2017. "Neural architecture search with reinforcement learning." *arXiv preprint* arXiv:1611.