

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI MATEMATICA «TULLIO LEVI-CIVITA»
CORSO DI LAUREA MAGISTRALE IN INFORMATICA

Leveraging image metadata for multi-label image classification

Relatore

Prof. LAMBERTO BALLAN

Controrelatore

Prof. ELISA RICCI, Università di Trento

TOBIA TESAN

This page intentionally left blank.

Abstract

In the context of multilabel image classification, the performance of visual classifiers can be boosted by using the implicit knowledge embedded in the similarity between the social network metadata of different images.

This approach is particularly effective for difficult images such as nonprototypical views of common objects and for rare classes.

We build upon the model of [JBFF15], which uses image metadata nonparametrically to generate neighbourhoods of related images according to Jaccard similarity, then uses a deep neural network to blend visual information from the image and its neighbours, allowing the model to perform well even when the vocabulary of metadata changes between training and testing.

Firstly, we propose variations on the model presented therein within the same general framework, using RNNs in the representation learning phase and semantic embeddings in the neighbourhood generation phase.

We perform comprehensive experiments on the NUS-WIDE dataset and show that our model outperforms that of [JBFF15].

Furthermore, we extend the general framework enabling it to leverage metadata explicitly with joint models; by using semantic embeddings to map metadata to a semantic space and decouple the parametric neural model from the low-level representation of metadata we achieve robustness to vocabulary change between training and testing.

We propose different models operating under this new general framework; we perform extensive experiments on NUS-WIDE with each model, characterizing their respective performances and showing that they outperform other models to a varying degree.

Finally, we propose an alternative and less biased experimental protocol for studies involving this sort of models.

This page intentionally left blank.

Contents

1	Introduction	9
1.1	Contribution of this thesis	10
1.2	Organization of this thesis	13
2	Background	15
2.1	Machine Learning	15
2.1.1	Classification problems	15
2.1.2	Inductive bias	16
2.1.3	Capacity, overfitting, underfitting	16
2.1.4	Parametric vs nonparametric models	18
2.2	Artificial neural networks	18
2.2.1	(Deep) feed-forward networks	20
2.2.2	Recurrent neural networks	23
2.2.3	Regularization techniques for deep neural networks	26
2.3	Image classification tasks and CNNs	28
2.3.1	Metrics for multi-label image classification	28
2.3.2	CNNs and Deep Learning	36
2.3.3	Transfer learning and pretrained models	38
2.4	Distributional word representations	40
3	Literature review	43
3.1	Metadata-based models	43
3.2	Neighbour-based models	43
3.3	Semantic models	46
4	Our models	47
4.1	Neighbourhood generation and size	47
4.2	Metadata semantic mappings π and distance measures δ	53
4.3	Neural architectures	54
4.3.1	Visual architectures	54
4.3.2	Joint architectures	55
5	Experiments	61
5.1	Experimental protocol	61
5.1.1	Dataset and pretrained weights	61
5.1.2	Semantic mappings and distance measures	62
5.2	Metrics	63

5.3	Results	64
5.3.1	Neural architectures	64
6	Variations upon the experimental protocol and future work	77
7	Conclusions	89

Notation

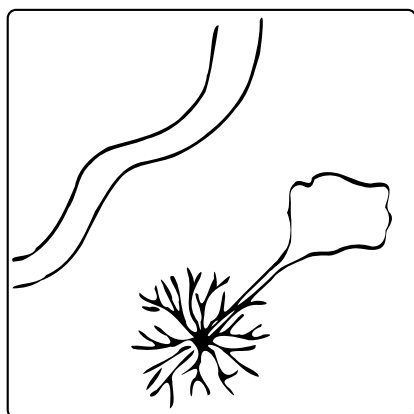
Throughout this document we use the following notation.

$a \triangleq b$	A is defined as b
$a \equiv b$	A is syntactically equivalent to b
$x^{(t)}$	x at time step t/t-th element of a succession
τ	Length of a discrete temporal sequence or succession
$x_{(i)}$	i-th element of vector \vec{x}
\mathcal{D}	Data-generating distribution
D_{train}	Training set
D_{test}	Test set
$D_{\text{validation}}$	Validation set
\vec{x}	A vector
x	An instance
y	Ground truth
\hat{y}	A prediction
\mathfrak{H}	Hypothesis space
$E[x]$	Expected value of x
$E_D[x]$	Expected value of x over distribution D
$E_s[x]$	Expected value of x over sample s
MSE_{train}	Mean squared (training) error
MSE_{test}	Mean squared (test) error
$Err_{\text{sample}(s)}$	Sample error
h	Hypothesis, or an approximator for some f^*
θ	Parameters of a model
w	Weight vector or weight matrix
b	Bias
$\sigma(x)$	An activation function over input x
$L_2(p)$	L_2 regularization
$L_1(p)$	L_1 regularization
λ	Regularization coefficient
η	Learning rate
\mathcal{L}	Loss
prec	Precision
rec	Recall
prec@k	Precision at k
rec@k	Recall at k
\mathcal{O}	Output units
\mathcal{I}	Input units
\mathcal{H}	Hidden units
\mathcal{U}	Units
x	An image x
$\phi(x)$	Visual features extracted from x
o_x	One-hot vector or binary vector for image x
$Z = \{z_1, z_2 \dots z_m\}$	Neighborhood

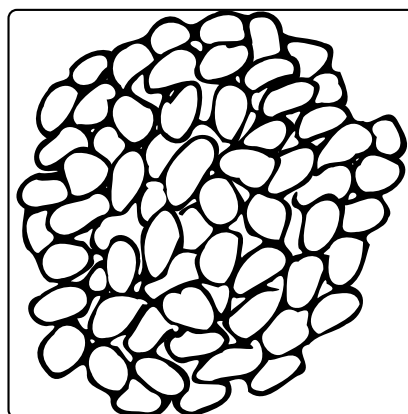
Z_x Neighborhood for image x
 δ Distance measure
 π Semantic mapping
 β Embedding dictionary
 $w_{net}(t)$ wnet representation for tag t
 $w_{2v}(t)$ w2v representation for tag t

Chapter 1

Introduction



(a) Mystery image one



(b) Mystery image two

Figure 1.1: Mystery images

The Oxford English Living Dictionary defines “*context*” as:

“The circumstances that form the setting for an event, statement, or idea, and in terms of which it can be fully understood” [oed]

That is, humans might possibly need some circumstantial information in order to achieve full understanding of the meaning of some sign or to correctly infer properties of an object.

Particularly, when identifying the content of an image, humans can benefit from the context that surrounds the image.

A bicycle tire can look very much an o-ring in a picture, which is to say, there exists a *sensory gap*:

Definition 1 (Sensory gap). The gap between the object in the world and the information in a (computational) description derived from a recording of that scene, due, for example, to clutter, occlusion... [SWS⁺01]

However while a picture of a round, slick, black object can be either, seeing it in a photograph of a car garage or of a watchmaker’s desk can help dissipate the ambiguity.

Even then, we might not be really sure about the *meaning* of the picture without some context – is a picture of a white dove a picture of a vertebrate, of bird, of something white, of a specific species of dove or maybe a symbol for peace – or maybe it’s a picture of interesting-looking clouds, occluded by a passing bird?

This is the *semantic* gap:

Definition 2 (Semantic gap). A lack of coincidence between the information that one can extract from the visual data and the interpretation that the same data have for a user in a given situation. [SWS⁺01]

One of the easiest forms of context to leverage in image classification is given by meta-data embedded in the image.

Consider for example a user of a photo sharing website who is shown the picture of figure 1.1b, or figure 1.1a, and asked to identify the object pictured.

He or she might have trouble readily identifying it.

However, if shown in the context of added metadata such as tags, such as in figure 1.3 the user might not have any trouble in understanding the subject as a bunch of pebbles.

Metadata other than tags, such as location or timestamp, might contain some *implicit* knowledge to tip off the user, such as in figure 1.2 – the user can more easily understand the image as an aerial view of a tree near the Arctic circle, in May, when the shadows are longer.

Figure 1.2 is an example of a “non-prototypical” view of a common object, which is ordinarily more difficult for a classifier to tackle. The prototypical, more common view of the same object – in fact, of the same scene – is that of figure 1.4.

However, it is not always trivial to *assign meaning* to metadata: in either cases this relies on the user knowing where Lapland is, or knowing the *meaning* of words such as “tree”, “rocks”, “pebbles”.

Consider now a non-Chinese speaking user faced with the screen of figure 1.5 – the available metadata might not be as useful in itself.

However, even then the user might be able to leverage it: if the user is cunning and familiar with how photo sharing sites generally work, he or she might with some luck click on the appropriate UI element and get to the screen of figure 1.6, i.e a gallery of images that share a tag with that of figure 1.5.

Now it would be once more clear, by looking at them, that we’re looking at a close up picture of a leaf.

Metadata and, particularly, metadata neighbours can be considerably effective in bridging the sensory and the semantic gap, as seen in the literature reviewed in chapter 3 and particularly in [JBFF15], [ML12], but there is more than one way to skin this particular cat.

1.1 Contribution of this thesis

The contribution of this thesis, building upon [JBFF15], is threefold: firstly, we propose two ways to enhance, within the same general framework, the performance of the model described therein, using RNNs and semantic embeddings.

Secondly, we propose a new general framework for joint models that leverages metadata explicitly rather than implicitly and we propose a way to achieve, nevertheless, robustness

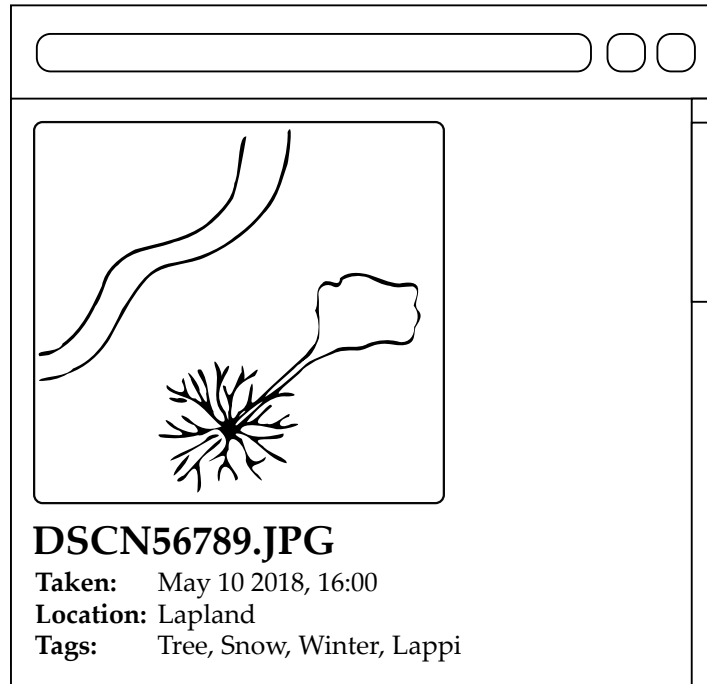


Figure 1.2: Mystery image one, plus metadata

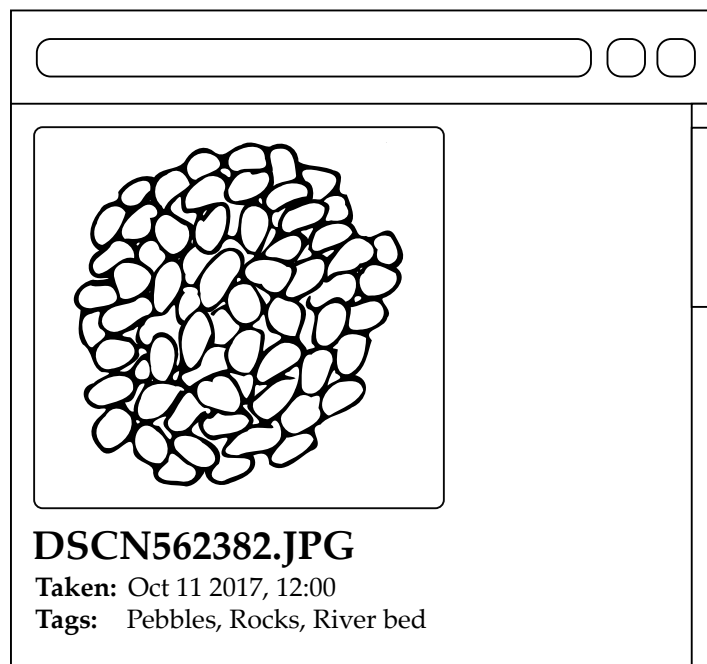


Figure 1.3: Mystery image two, plus metadata

to vocabulary change between training and testing through the use of semantic embeddings.

We describe a selection of models that implement it; we characterize the performance

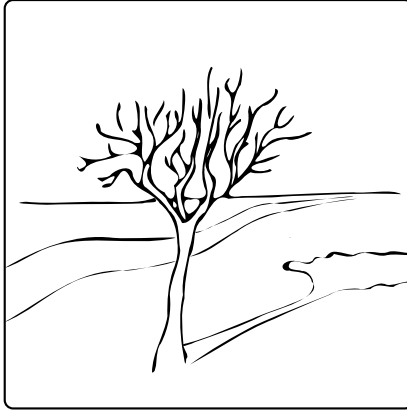


Figure 1.4: Prototypical view of a tree; same scene as figure 1.1a, different perspective



Figure 1.5: Mystery image

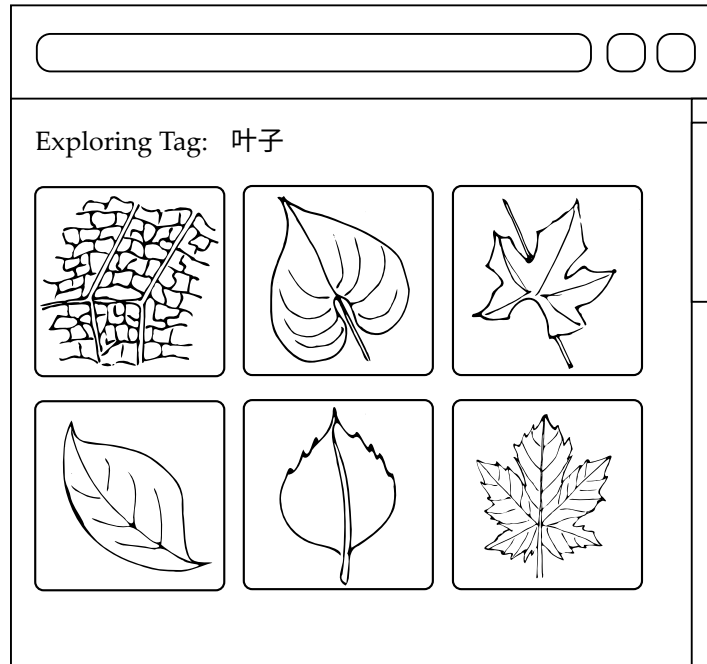


Figure 1.6: A web page displaying the mystery image of figure 1.5 in the context of other images tagged with a same tag

of all models through comprehensive experiments on the NUS-WIDE dataset.

Finally, we propose what we believe is an ideal experimental protocol to be used for this sort of relatively new family of models.

1.2 Organization of this thesis

In chapter 2 we provide some essential background.

Section 2.1 deals with the generalities of machine learning and, specifically, (deep) neural networks.

In section 2.2 we give some background on image annotation and, particularly, the task of multi-label object identification.

In Chapter 3 we review existing literature with a focus on some recent work.

In Chapter 4 we discuss our framework and models.

Specifically, in section 4.2 we discuss semantic mappings and distance measures we use to identify neighbourhoods; in section 4.3 we discuss neural architectures.

In chapter 5 we present the experiments we carried out; in section 5.1 we detail the experimental protocol used; in particular in section 5.1.1 we discuss the dataset and pre-trained weights used; in section 5.2 we present the evaluation metrics used and give upper bounds on our dataset; finally, we present and extensively discuss the experimental results in section 5.3.

We discuss alternate experimental protocols in chapter 6.

We draw our conclusions in chapter 7.

This page intentionally left blank.

Chapter 2

Background

2.1 Machine Learning

Machine learning is a particularly effective technique when [Mit97] large sets of data are available that contain implicit regularities that can be discovered automatically, explicit algorithmic knowledge is lacking, and/or there is a need for dynamic adaptation to changing conditions.

One general definition of machine learning is found in [Mit97]:

Definition 3. A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .

Machine learning algorithms can be broadly organized in the following taxa [GBC16]:

- *Supervised learning* algorithms, which experience a dataset where every example is associated with a label or target value
- *Unsupervised learning* algorithms, which experience a dataset containing many features, then learn useful properties of the structure of this dataset.
- *Reinforcement learning* algorithms, which interact with an environment rather than *passively* experiencing a dataset.

2.1.1 Classification problems

The task of classification, along with the closely related one of regression, is among the most popular supervised learning tasks.

Consider a distribution \mathcal{D} of pairs of *instances* and labels $\langle x, y \rangle \in (\mathbb{R}^n \times [0, 1])$ – for example a vector of vital parameters and a label signifying the presence or absence of a certain disease in patients.

In other words, suppose there exists a relation ¹ \mathcal{R} s.t.

$$\mathcal{R}(x, y)$$

¹Not necessarily univalent: when collecting data in the real world it is possible that two patients have the same exact vital parameters but end up with different diagnoses

for all $\langle x, y \rangle$ in \mathcal{D} .

Suppose we have access to a finite set $D_{\text{train}} \subseteq \mathcal{D}$ of discrete *examples* $\langle x_i, y_i \rangle$, called *training set*, that is ideally representative of \mathcal{D} , in which the right-hand side of the pair is called *ground truth*.

Classification amounts to *generalizing* from said examples in order to derive (*learn*) a function

$$h : \mathbb{R}^n \rightarrow \{0, 1\}$$

called “hypothesis”, chosen from an hypothesis space

$$\mathfrak{H} \subseteq (\mathbb{R}^n \rightarrow \{0, 1\})$$

such that it approximates \mathcal{R} .

The ability to perform well on previously unobserved inputs is called “generalization”.

In multiclass classification, we have that $\langle x_{(i)}, y_{(i)} \rangle \in (\mathbb{R}^n \times [0, 1, \dots, k])$ and in multilabel classification we have $\langle x_{(i)}, y_{(i)} \rangle \in (\mathbb{R}^n \times [0, 1]^k)$ – it can either be the case that the instance belongs to a given class or not.

2.1.2 Inductive bias

Generalizing beyond the training examples *mandates* that the machine learning system contains some form of inductive bias – i.e. some further assumption for choosing a generalization over another.

Without such assumptions, it is impossible to go beyond trivial rote learning of the training examples [Mit97]; such assumptions are dependent on our knowledge of the task and the domain.

Intuitively, the popular connect-the-dots puzzle is solved by making the *assumption* that the dots are to be connected with straight lines: one could imagine a variation upon the classic puzzle in which dots are to be connected with parabolas (three by three, of course) or with a single spiral, or more dots.

Without such assumptions, the solver would not be able to do anything with the dots.

Note, moreover, that according to the **no free lunch theorem** [WM97], no machine learning algorithm is *universally* any better than any other: averaged over *all* possible data-generating distributions, every classification algorithm has the same average performance when classifying previously unobserved instances.

We will therefore seek an inductive bias that is practically effective for our task, domain and dataset.

Inductive bias imposes a representation for the learner which allows some degrees of freedom and amounts to choosing the set of functions that it can possibly learn, called the *hypothesis space*.

2.1.3 Capacity, overfitting, underfitting

Definition 4 (Generalization error). Generalization error of a classifier h is the expected value of its error on a new input x extracted from the distribution of possible inputs \mathcal{D}

$$\text{Err}_{\mathcal{D}}(h) \triangleq \mathbb{E}_{\langle x, y \rangle \in \mathcal{D}} [\delta(y, h(x))]$$

where

$$\delta(y, y') = \begin{cases} 1 & \text{if } y = y' \\ 0 & \text{otherwise} \end{cases}$$

It is also known as *test error*.

One way of approximating it is by applying the classifier over a test set $D_{\text{test}} \subseteq \mathcal{D}$ of examples collected separately from the training set and computing the mean square error [GBC16]:

$$\text{MSE}_{\text{test}} \triangleq \frac{1}{2} \sum_{(x,y) \in D_{\text{test}}} (y - h(x))^2 \quad (2.1)$$

This measure is a form of sample error in that it is defined over a specific set of examples instead of a distribution [Mit97].

Assume now we we have a training set D_{train} and a test set extracted from the space of examples in an i.i.d. fashion.

Similarly, we can approximate

$$\mathbb{E}_{x \in D_{\text{train}}} [\delta(y, h(x))]$$

with

$$\text{MSE}_{\text{test}} \triangleq \frac{1}{2} \sum_{(x,y) \in D_{\text{test}}} (y - h(x))^2 \quad (2.2)$$

The expected error of an arbitrary classifier h over each set is exactly the same.

$$\text{Err}_{D_{\text{test}}} h = \text{Err}_{D_{\text{train}}} h$$

However, if the classifier is not “arbitrary” but instead is refined in order to minimize the error over D_{train} , so it must be that the expected error over the test set is bounded below by the expected value of the error on the training set:

$$\text{Err}_{D_{\text{test}}} h \geq \text{Err}_{D_{\text{train}}} h \quad (2.3)$$

And also

$$\text{Err}_{\mathcal{D}} h \geq \text{Err}_{D_{\text{train}}} h \quad (2.4)$$

Because of inequality equation (2.4), in order to minimize $\text{Err}_{\mathcal{D}} h$, it is desirable

1. To make $\text{Err}_{D_{\text{train}}} h$ – or its approximant, e.g. $\text{Err}_{D_{\text{train}}} h$ – small
2. To make the bound tight

When the first condition is not met, we have underfitting; when the second condition is not met, we have overfitting.

More formally²:

²Note that underfitting is *not* symmetrical to this definition: it is what happens when $\text{Err}_{D_{\text{train}}} h$ can't be made sufficiently small.

Definition 5 (Overfitting). Given a hypothesis space \mathcal{H} a hypothesis $h \in \mathcal{H}$ is said to overfit the data iff $\exists h' \in \mathcal{H}$ such that

$$\text{Err}_{\text{sample}}(h) < \text{Err}_{\text{sample}}(h')$$

but

$$\text{Err}_{\mathcal{D}}(h) > \text{Err}_{\mathcal{D}}(h')$$

i.e. h has a smaller error over the training examples, but h' has a smaller error over the entire distribution of instances [Mit97].

Both are a function of a model's capacity: a model with low capacity might tend to underfit; a model with high capacity might overfit unless precautions are taken.

One way to control the capacity of a learning algorithm is by choosing its hypothesis space, i.e. the set of functions that the learning algorithm is allowed to select as a the solution [GBC16].

2.1.4 Parametric vs nonparametric models

Definition 6 (Parametric learning). A learning model that summarizes data with a set of parameters of fixed size – i.e. learns a function described by a parameter vector whose size is finite and fixed – is called a parametric model [GBC16] [RN16].

Neural Networks and Support Vector Machines are paradigmatic examples of parametric models.

Those models that are not parametric are called *instance-based models* or simply “non-parametric models” [RN16].

One example of such an algorithm is nearest neighbor regression. A nearest neighbor regression model simply stores the instances from the training set and, when queried, it will look up the nearest entry in the training set and return the associated regression target.

Their complexity is therefore a function of the training set size [GBC16].

Hyperparameter optimization

Sometimes we wish to tune some higher-order parameters or *hyperparameters*, that control the behaviour and performance of *the learning algorithm* but are not part of the learned model.

We shall therefore wish to equip ourselves with a third set, called “validation set” that will enable us to compare different models and choices of hyperparameters in an unbiased way.

2.2 Artificial neural networks

One very popular family of machine learning models is that of Artificial Neural Networks, or ANNs.

ANNs are vaguely inspired by (a 1950s understanding of) the animal brain and are especially suited to classification and regression tasks with noisy sampled data, such as photographs and videos.

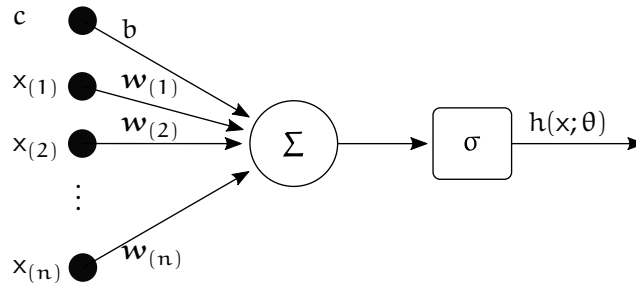


Figure 2.1: Perceptron

The most ubiquitous sort of ANN is based on the perceptron, which constitutes in itself the simplest case of an artificial neural network.

The computational activity of a single perceptron amounts to computing

$$h(\mathbf{x}; \theta) = f(\mathbf{x}; \mathbf{w}, b) = \sigma(\mathbf{x}^T \mathbf{w} + b) \quad (2.5)$$

for some $\mathbf{x} \in \mathbb{R}^n$ and some $\mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}$ and an *activation function* $\sigma: \mathbb{R} \rightarrow \mathbb{R}$.

Traditionally – and for our purposes until section 2.2.1 – , let $\sigma = \text{sign}$.

This is laid out in figure 2.1.

Training the perceptron means choosing “adequate” values for the *learnable parameters* $\theta = \langle \mathbf{w}, b \rangle$ in order to best approximate \mathcal{D} .

Gradient descent

Gradient-based optimization techniques are by far the most common way to obtain θ .

The idea behind applying gradient-based optimization to neural networks is to use it to update the weights in a way such that they move in the direction that minimizes a given *loss function* \mathcal{L} which we expect to be a good predictor of $\text{Err}_{\mathcal{D}}(h(\mathbf{x}, \theta))$.

In general, to find a choice of vector \vec{x} that minimizes function $f(\vec{x})$ for an arbitrary function f , gradient descent algorithms iteratively compute

$$\mathbf{x}^{(i)} := \vec{x}^{(i-1)} - \eta \nabla f(\vec{x}^{(i-1)})$$

where

$$\nabla f(\vec{x}) \triangleq \left[\frac{\delta f}{\delta x_{(1)}}, \frac{\delta f}{\delta x_{(2)}}, \dots, \frac{\delta f}{\delta x_{(n)}} \right]$$

By using gradient descent to minimize the loss function \mathcal{L} we then maximize the performance of our model – or minimizing the expected error.

\mathcal{L} often is related or is a function of the $\text{Err}_{\text{sample}}(h(\mathbf{x}, \theta))$, but may also include additional terms, such as *regularization terms*, which we discuss in section 2.2.3 and serve to limit the capacity of the model and therefore contrast overfitting.

Since gradient descent “chases” the direction of the gradient, it has the potential of getting “stuck” in **critical points** in which the gradient is 0-valued, such as **local minima**, **local maxima** or **saddle points** [GBC16].

In practice, gradient descent empirically seems to be able to avoid local maxima and escape saddle points in many cases [GBC16]; while local minima constitute an open research

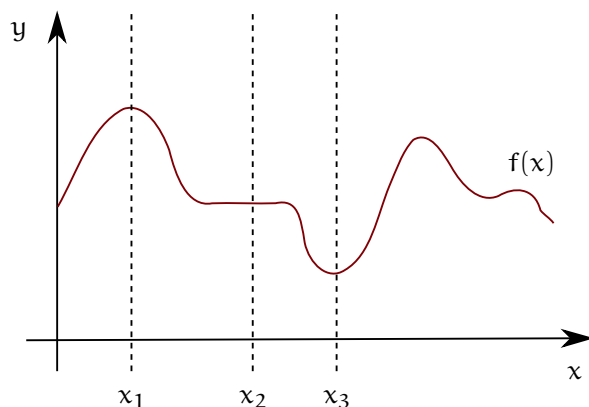


Figure 2.2: Types of critical points: x_1 is a local minimum, x_2 is a saddle point, x_3 is a saddle point

area, recent findings suggest that most local minima in practical networks have more often than not an acceptably low value, and are thus not the most likely cause of difficulties in optimization [GBC16]. There are several families of gradient descent algorithms that afford different trade-offs between the accuracy of the parameter update and the time it takes to perform an update.

1. Batch gradient descent computes the gradient of the cost function w.r.t. to the parameters for the entire training dataset, as seen in listing 1. it is therefore expensive in terms of space and time.

The error measure is arbitrary, but a popular choice [Mit97] is to define $\text{Err}(\theta)$ to be the MSE as defined in equation (2.2), possibly with the addition of regularization terms.

2. Stochastic gradient descent, as seen in listing 2, approximates batch gradient descent by performing a parameter update for each training example. Stochastic gradient descent has a tendency to fluctuate, which can help with local minima, but ultimately can complicate convergence.
3. Mini-batch gradient descent is a compromise between the two; it performs an update for every mini-batch of n training examples. This reduces fluctuation and can lead to more stable convergence; it lends itself particularly well to implementation on GPUs and CPUs with vector instruction sets such as Intel AVX.

2.2.1 (Deep) feed-forward networks

Perceptrons can be composed together as *units* to form larger neural networks, thus increasing the achievable computational power (in a computability theory sense).

Definition 7 (Neural network architecture). The overall structure of a neural networks, given by how many units make it up and how they are interconnected, is called “architecture”.

The architecture of a network is described by a directed graph such as that of figure 2.3, in which there is one node for every unit and one edge for every connection between the output of a unit and the input of another.

GRADIENT-DESCENT(D_{train}, η)

```
1 //  $\theta \triangleq [\mathbf{w} \ b] \in \mathbb{R}^{n+1}$  are the weight vectors to be optimized
2 //  $D_{\text{train}}$  are the training examples
3 //  $\eta$  is the learning rate
4 for  $i = 1$  to  $n + 1$ 
5      $\theta_{(i)} := \text{RANDOM}(0, 1) * \epsilon$ 
6 while  $\text{Err}(\theta) > \epsilon$  // Variation: until a fixed number of iterations is reached
7     // One epoch
8     for  $i = 1$  to  $n$ 
9          $\Delta\theta_{(i)} := 0$ 
10        for each  $\langle x, y \rangle \in D_{\text{train}}$ 
11             $o := \sigma(\mathbf{w}x + b)$ 
12            for  $i = 1$  to  $n$ 
13                 $\Delta\mathbf{w}_{(i)} := \Delta\mathbf{w}_{(i)} + \eta(y - o)x_i$ 
14                 $\Delta b := \Delta b + \eta(y - o)$ 
15            //  $\Delta\theta \approx -\eta\nabla\theta$ ; see e.g. [Mit97] for proof
16            for  $i = 1$  to  $n$ 
17                 $\theta_{(i)} := \theta_{(i)} + \Delta\theta_{(i)}$ 
18 return  $\theta$ 
```

Listing 1: Gradient descent

STOCHASTIC-GRADIENT-DESCENT(D_{train}, η)

```
1 //  $\theta \triangleq [\mathbf{w} \ b] \in \mathbb{R}^{n+1}$  are the weight vectors to be optimized
2 //  $D_{\text{train}}$  are the training examples
3 //  $\eta$  is the learning rate
4 for  $i = 1$  to  $n$ 
5      $\theta_{(i)} := \text{RANDOM}(0, 1) * \epsilon$ 
6 while  $\text{Err}(\theta) > \epsilon$  // Variation: until a fixed number of iterations is reached
7     // One epoch
8     for each  $\langle x, y \rangle \in D_{\text{train}}$ 
9          $o := \sigma(\mathbf{w}x + b)$ 
10        for  $i = 1$  to  $n$ 
11             $\mathbf{w}_{(i)} := \mathbf{w}_{(i)} + \eta(y - o)x_i$ 
12             $b := b + \eta(y - o)$ 
13 return  $\theta$ 
```

Listing 2: Stochastic gradient descent

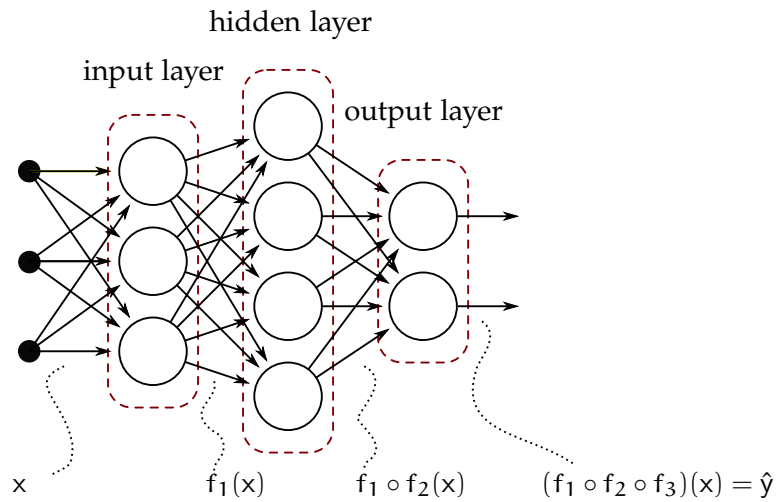


Figure 2.3: A multilayer perceptron

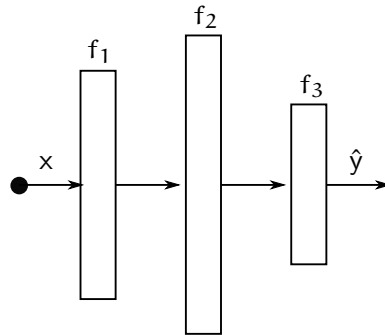


Figure 2.4: Alternate graphical representation of the network of figure 2.3

Commonly, networks are organized into groups called “layers” such that the output of each layer is a function of the previous one’s activation.

Consider for example figure 2.3: it is the case that

$$\hat{y} = (f_3 \circ f_2 \circ f_1)(x) \tag{2.6}$$

Where $f_{1\dots 3}$ is the function $\mathbb{R}^n \rightarrow \mathbb{R}^m$ computed by each layer.

Often in literature architectures are described graphically simply by their layers, such as in figure 2.4.

The most straightforward architecture is the “deep feedforward network” or multilayer perceptron.

Unlike recurrent networks, discussed in section 2.2.2 there is no feedback connections in this sort of network and, therefore, the output of every layer is defined in terms of the output yielded by preceding layer or, less commonly (e.g. [HZRS16]) of more than one previous layer.

In this sense, the structure of a feedforward network can be thought of as a directed *acyclic* graph.

The layers in an MLP are further subdivided into input layer, hidden layers (of which figure 2.3 only has one) and output layer.

MLPs almost always incorporate units with non-linear activation function σ , that endow them with the ability, given enough hidden units, to approximate the set of functions described in the **universal approximation theorem**, which amount to the set of continuous functions on a closed and bounded subset of \mathbb{R}^n [GBC16].

In contrast, the simple perceptron equipped with a linear activation function famously cannot compute the XOR function [MP72].

Note, however, that the universal approximation theorem says nothing about the possibility of *learning* those functions: it might very well be the case that any training algorithm could fail to converge to the appropriate weights.

Backpropagation

When training a multilayer network, the gradient descent algorithm of listing 1 does not suffice.

The most glaring difference wrt the context of the single-neuron network of figure 2.1 is that D_{train} provides target values y only for network outputs, so the the error of the hidden units must be imputed somehow: this is the **credit assignment problem**.

The hypothesis space of a multilayer perceptron with several output units \mathcal{O} is larger than a perceptron with a single output and defined over all possible weight values for all units; the error is defined as ³:

$$\text{Err}(\theta) = \frac{1}{2} \sum_{d=\langle x,y \rangle \in D_{\text{train}}} \sum_i (y_{(i)} - h_{(i)}(x; \theta))^2$$

The algorithm presented in listing 3 optimizes the parameters θ wrt the squared error between the output values of the network and the target values for these outputs.

One complete pass over the training set – i.e. one iteration of the outermost loop – is called an “epoch”.

An in-depth explanation and proof can be found in [Mit97]; an alternative one, based on a graph-theoretical intuition, can be found in [Roj96] that also applies to architectures other than the MLP.

2.2.2 Recurrent neural networks

Unlike the feed-forward networks described in section 2.2.1, RNNs allow for recurrent connections.

Which is to say, if the network is represented as a graph in which each unit corresponds to a node which has a directed edge towards any other unit its output is a function of, this graph is no longer a DAG but allows for cycles or loops.

The presence of feedback loops makes the network stateful, and its output $\hat{y}^{(t)}$ at time step t is a function of the network’s input $x^{(t)}$ at time step t *and* its hidden internal state at time step $t - 1$.

³Or, equivalently [Mit97]:

$$\text{Err}(\theta) = \frac{1}{2} \sum_{d=\langle x,y \rangle \in D_{\text{train}}} \sum_i (y_{(i)} - \hat{y}_{(i)})^2$$

where $o_{(i)}$ is the output of the i -th output unit.

```

BACK-PROPAGATION( $D_{\text{train}}, \mathcal{O}, \mathcal{H}, \mathcal{J}$ )
1 //  $\mathcal{J} = \{i_1 \dots i_m\}$  are the input units
2 //  $\mathcal{O} = \{o_1 \dots o_o\}$  are the output units
3 //  $\mathcal{H} = \{h_1 \dots h_p\}$  are the hidden units
4 //  $\theta_u = [\mathbf{w}_u \ b_u]$  is the weight vector associated with each unit
5 // We denote the function of each unit  $u \in \mathcal{U} \cup \mathcal{O}$  as  $f_u(\vec{x}; \theta_u)$ 
6 //  $D_{\text{train}}$  are the training examples
7 //  $\delta_u$  is error term for the  $u \in \mathcal{J} \cup \mathcal{O}$ , which serves to impute  $u$ 's share of the error
8  $w_u := \text{RANDOM}(0, 1) \cdot \varepsilon \ \forall u \in \mathcal{O} \cup \mathcal{H}$ 
9 while  $\text{Err}(\theta) > \varepsilon$ 
10 // One epoch
11 for each  $\langle x, y \rangle \in D_{\text{train}}$ 
12 // Phase 1: Propagate forward by computing the output for each unit
13 for  $i = 1$  to  $m$ 
14  $i_{(i)} := x_{(i)}$ 
15 for  $i = 1$  to  $p$ 
16  $h_{(i)} := f_{h_{(i)}}(i_{(1)} \dots i_{(m)}; \theta_{h_{(i)}})$ 
17 for  $i = 1$  to  $o$ 
18  $o_{(i)} := f_{o_{(i)}}(h_{(1)} \dots h_{(m)}; \theta_{o_{(i)}})$ 
19 // Phase 2: Propagate and impute error backwards
20 for  $i = 1$  to  $m$ 
21 // For each output unit  $o$ , compute its error term  $\delta_o$ 
22  $\delta_{o_{(i)}} := (o_{(i)}(1 - o_{(i)}))(y_{(i)} - o_{(i)})$ 
23 for  $j = 1$  to  $p$ 
24 // For each hidden unit  $h$ , compute its error term  $\delta_h$ 
25  $\delta_{h_{(j)}} := (o_{(j)}(1 - o_{(j)})) \sum_{k=1}^o (\mathbf{w}_{(k,j)} \delta_{o_{(k)}})$ 
26 //  $\mathbf{w}_{(k,j)} \in \mathbb{R}$  is the  $j$ -th element of the weight vector for unit  $k$ 
27 for each  $w_{(i,j)}$ 
28  $\mathbf{w}_{ij} := \mathbf{w}_{ij} + \Delta \mathbf{w}_{ij}$ 
29  $b_{ij} := b_{ij} + \Delta b_{ij}$ 
30 return  $\theta$ 

```

where

$$\Delta \mathbf{w}_{ij} = \eta \delta_i x_{ji}$$

$$\Delta b_{ij} = \eta \delta_i$$

Listing 3: Backpropagation for a three-layer network [Mit97]

This results in the computing capability of RNNs to be broadened –in fact, it has been shown that recursive neural networks are Turing-complete [DS02], [SS95].

In practice, RNNs are often used and particularly well-suited for time-series and data with an otherwise sequence: the “past” sequential information is preserved in the network’s current state.

Consider the network of figure figure 2.5a.

To simplify notation, we will now refer to the weights of whole layers as weight matrices, instead of single units, after [GBC16]; moreover, only for for the remainder of this section, will draw entire layers as nodes in a graph, to reduce visual clutter.

Let \mathbf{u}, \mathbf{w} and \mathbf{v} be weight matrices. The network output $\hat{y}^{(t)}$ any given time step t is then

$$\mathbf{a}^{(t)} = \mathbf{u}\mathbf{x}^{(t)} + \mathbf{w}\mathbf{h}^{(t-1)} + \mathbf{b} \quad (2.7)$$

$$\mathbf{h}^{(t)} = \sigma_{\mathbf{h}}(\mathbf{a}^{(t)}) \quad (2.8)$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{v}\mathbf{h}^{(t)} \quad (2.9)$$

$\sigma_{\mathbf{h}}$ is an activation function; \tanh is a popular choice.

Do note that if the network were connected as in figure 2.5b, it would be strictly less powerful (i.e. not Turing-complete) [GBC16].

Such a network can be trained to predict a sequence $\hat{y}^{(1)}, \hat{y}^{(2)} \dots \hat{y}^{(\tau)}$ of τ output vectors (of which only a subsequence $\hat{y}^{(m)} \dots \hat{y}^{(\tau)}$ may actually be of interest) as a function of an input sequence $x^{(1)}, x^{(2)}, \dots x^{(\tau)}$.

Backpropagation through time

Computing the error gradient through a recurrent neural network implies applying back-propagation to an **unrolled** version of the network, as in figure 2.7, with one “replica” for each element of the training sequence (or time step).

Gradients obtained by back-propagation may then be used with any general-purpose gradient- based techniques to train an RNN.

Let τ be the length of a training instance; then the runtime is $O(\tau)$ – this cannot be reduced by parallelization because the forward propagation graph is inherently sequential – as is the memory cost [GBC16].

The back-propagation algorithm applied to the unrolled graph is called back-propagation through time (BPTT).

The unrolled graph of figure 2.7 has a weight vector \mathbf{w} shared across multiple connections that feed into another. equation (2.8) implies that in figure 2.7

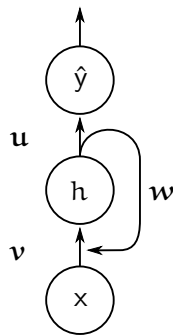
$$\mathbf{h}^{(t)} = \mathbf{u}\mathbf{x}^{(t)} + \mathbf{w}(\mathbf{u}\mathbf{x}^{(t-1)} + \mathbf{w}(\mathbf{u}\mathbf{x}^{(t-2)} + \mathbf{w}(\dots)))$$

and by left distributivity

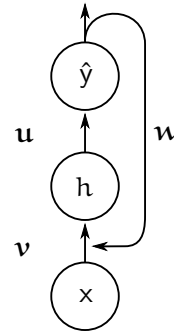
$$\mathbf{h}^{(t)} = \mathbf{u}\mathbf{x}^{(t)} + \mathbf{w}(\mathbf{u}\mathbf{x}^{(t-1)}) + \mathbf{w}^2(\mathbf{u}\mathbf{x}^{(t-2)}) + \dots + \mathbf{w}^{(\tau)}(\mathbf{v}(\mathbf{x}))$$

Those $\mathbf{w}^{(1)} \dots \mathbf{w}^{(\tau)}$ terms with large exponents mean that, depending on the value of \mathbf{w} the gradient can get extremely small or extremely large, which makes the training respectively extremely slow to converge or highly unstable.

This is known as the **vanishing** resp. **exploding gradient problem**.



(a) A recurrent neural network



(b) A strictly less powerful (“bad”!) recurrent neural network

Do note that this is a problem with the training process, not with the representational capacity of the network.

Gated RNNs, such as LSTM and leaky units have been developed to counter the vanishing gradient problem [GBC16].

At the core of LSTM is the notion of different neural networks acting as *gates* that, depending on context, dynamically control the weight of the recurrent loop, with the effect of selectively adding or removing information from the cell state.

Recently GRUs have emerged [CVMG⁺14] that afford most of the same advances while being easier to train due to a lower number of parameters.

Gated units have been found extremely successful and have achieved widespread popularity for different applications [GBC16].

Clipping [PMB13] has been proposed as another technique to control the vanishing and exploding gradient problem.

At the most basic level, a pre-determined gradient threshold is introduced; then, gradients whose norm exceeds this threshold are “clipped” to match it; several variations on this basic technique exist.

2.2.3 Regularization techniques for deep neural networks

Dropout has relatively recently been proposed as a technique to counter overfitting [SHK⁺14]. At the core is the observation that with limited training data, neurons will co-adapt to learn complex relationships that are entirely the result of sampling noise; randomly turning off select neurons during training amounts to sampling from a number of “subnetworks”, preventing coadaptation. The result is shown graphically in figure 2.6.

Adding a **regularization term** to the loss function \mathcal{L} that imposes a penalty on large weights is a popular way to control overfitting; it has philosophical implications that go back to Occam’s razor.

L_1 and L_2 -regularization are popular; they both amount to adding to the loss function a coefficient, multiplied by some λ constant.

In L_1 regularization, we add $\lambda L_1(\Theta)$, where $L_2(\Theta) \triangleq \frac{1}{2} \|w\|_2^2$, whereas in L_2 regularization $L_1(\Theta) = \|w\|_1$.

L_1 regularization has a tendency to remove some dimensions altogether and is thus useful for feature selection.

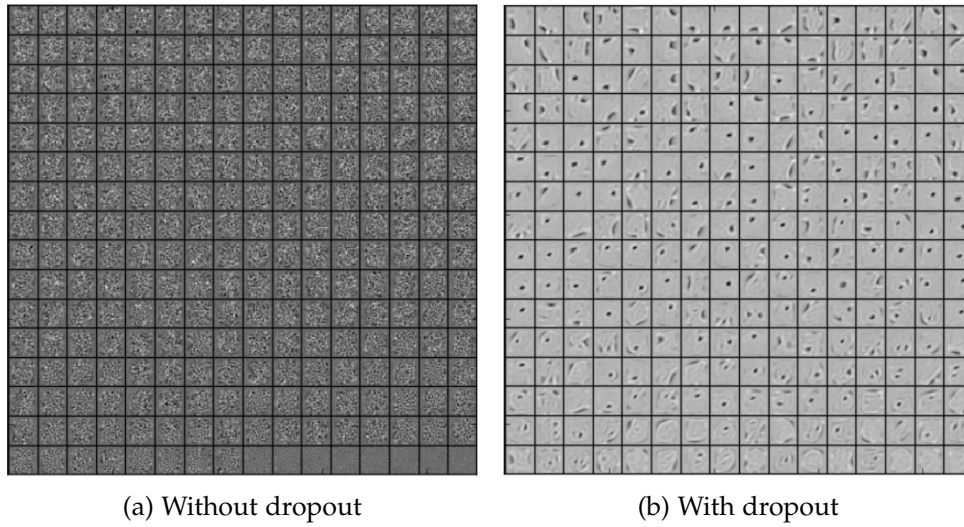


Figure 2.6: Features learned on MNIST with one hidden layer autoencoders having 256 rectified linear units, with and without dropout. Image reproduced from [SHK⁺14]

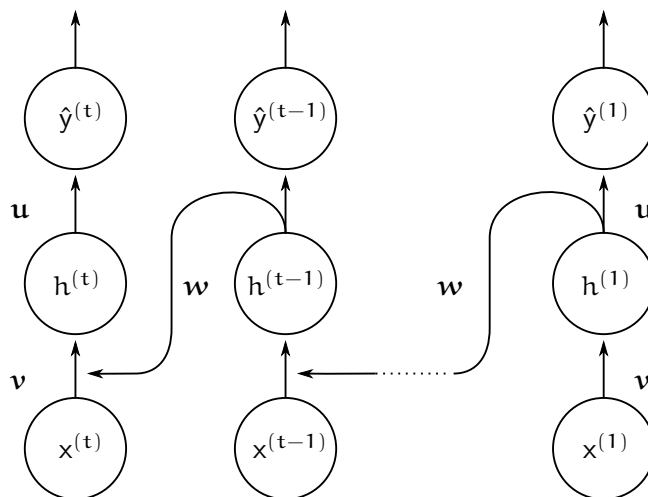


Figure 2.7: Unrolled RNN of figure 2.5a

2.3 Image classification tasks and CNNs

Image classification is the task of assigning an input image one label from a fixed set of categories or, equivalently, the task of categorizing images into one of several predefined classes.

Image classification is one of the core problems in the field of computer vision: it has a large variety of practical applications and many seemingly distinct computer vision tasks (such as object detection, segmentation) can be reduced to image classification [cs219].

Challenges in image classification include viewpoint variation, scale variation, deformation, occlusion, clutter and intraclass variation [cs219].

Clearly, the task of image classification can be easily cast as a possibly multi-class or multi-label classification problem: we will take a special interest in CNNs for image classification, which we will delve into in section 2.3.2 and which are a form of feed-forward multi-layer neural networks.

We shall first discuss the more general problem of evaluating an image classifier's performance; we will limit ourselves to multi-label classification.

2.3.1 Metrics for multi-label image classification

Confusion matrix Recall that in a multi-label classification problem, given x , we wish to predict, with a function h , y in pairs in

$$\langle x, y \rangle \in (\mathbb{R}^n \times [0, 1]^m)$$

drawn from a distribution \mathcal{D} .

Suppose, then, that we learn

$$h : \mathbb{R}^n \rightarrow \{0, 1\}^m$$

For each instance x in a pair $\langle x, y \rangle$, our classifier will then yield an m -dimensional vector \hat{y} in which each entry signifies the predicted presence or absence of a given label – in other words, our classifier can be thought of as a battery of m binary classifiers.

There are then 4 possible outcomes for every i -th label:

1. $y_{(i)} = \hat{y}_{(i)} = 1$. This outcome is called a *true positive*.
2. $y_{(i)} = \hat{y}_{(i)} = 0$. This outcome is called a *true negative*.
3. $y_{(i)} = 0 \wedge \hat{y}_{(i)} = 1$. This is called a *false positive*, also known as “type I error”.
4. $y_{(i)} = 1 \wedge \hat{y}_{(i)} = 0$. This is called a *false negative*, also known as “type II error”.

This is depicted graphically in figure 2.8.

The 4×4 *confusion matrix* for each instance (resp. label) is obtained by counting the number of labels (resp. instance) for each outcome.

An example is found in table 2.1.

x	y	\hat{y}	outcome
1	1	1	TP
2	0	0	TN
3	1	1	TP
4	1	0	FN
5	0	1	FP
6	1	1	TP
7	0	1	FP
8	0	0	TN

(a) Classification results

	pos	neg
true	3	2
false	2	1

(b) Confusion matrix for table 2.1a

Table 2.1: Computing the confusion matrix: an example

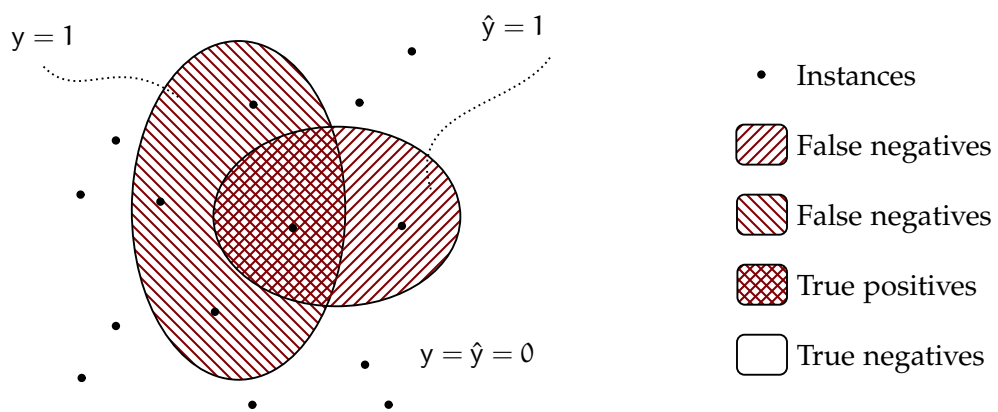


Figure 2.8: True positive, false positives, true negatives and false negatives

Precision and recall

From the confusion matrix we can synthesize a few useful metrics.

Let p be a pair $\langle y, \hat{y} \rangle$ of a ground truth vector y for an instance and a predicted vector \hat{y} .

Let then $TP(p), FP(p), FN(p), TN(p)$ be respectively the number of true positives, false positives, true negatives, false negatives in p , i.e:

$$TP(\langle y, \hat{y} \rangle) = |\{i : y_{(i)} \cdot \hat{y}_{(i)} = 1\}|$$

$$FP(\langle y, \hat{y} \rangle) = |\{i : \hat{y}_{(i)} - y_{(i)} = 1\}|$$

$$TN(\langle y, \hat{y} \rangle) = |\{i : y_{(i)} \cdot \hat{y}_{(i)} = 0\}|$$

$$FN(\langle y, \hat{y} \rangle) = |\{i : y_{(i)} - \hat{y}_{(i)} = 1\}|$$

Precision intuitively indicates how trustworthy a positive prediction is; it is defined as:

Definition 8 (Precision).

$$\text{prec}(p) \triangleq \frac{TP(p)}{TP(p) + FN(p)}$$

Notice that it is not defined when the set of positive instances in a given set is the empty set.

Precision can be interpreted [MRS08] as:

$$P(\text{positive instance} \mid \text{positive prediction})$$

Conversely, recall – also known as “sensitivity” – intuitively indicates how exhaustive positive predictions are:

Definition 9 (Recall).

$$\text{rec}(p) \triangleq \frac{TP(p)}{TP(p) + FP(p)}$$

Notice that it is not defined when the number of positive predictions is zero.

Recall can be interpreted [MRS08] as:

$$P(\text{positive prediction} \mid \text{positive instance})$$

A trade-off is implied between precision and recall: trivially, a classifier that always gives a positive prediction will have a recall of 1 and a small precision ϵ and viceversa.

It is the “wiggle room” between these two metrics that determines the performance of a system: an “omniscient” black box could achieve precision *and* recall of 1.

Suppose now, as it can easily be the case with ANNs⁴, that our approximator has type

$$f : \mathbb{R}^n \rightarrow [0, 1]^m$$

i.e. it outputs some number $E[y_{(i)}] \in [0, 1]$ for every i -th class out of m .

Those can be interpreted as the confidence level or “degree of belief” that a given instance belongs to a certain class.

What constitutes a “positive” prediction then?

⁴But is not necessarily the case with e.g. decision trees

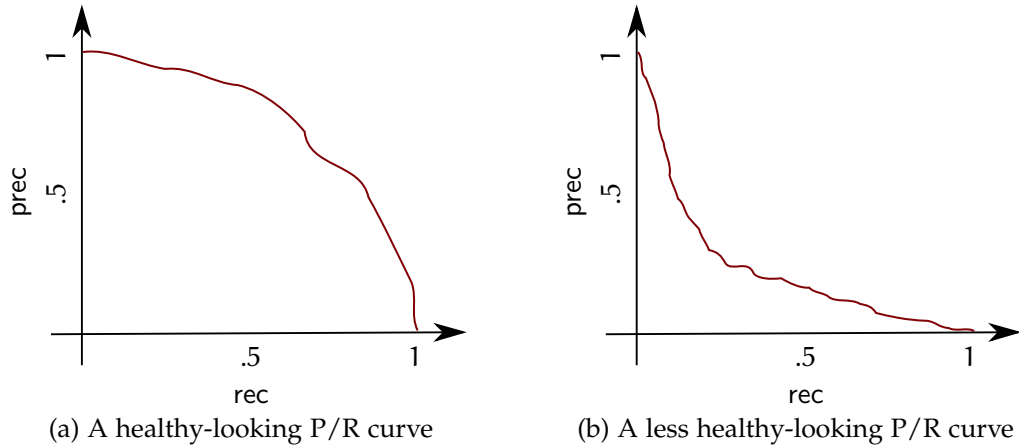


Figure 2.9: Examples of precision/recall curves

Taking a cue from the field of information retrieval we can then rank them according to the ordering \geq and consider the i -th label to be a positive prediction iff it is among the top k -ranking labels for some arbitrary k .

This is the definition of precision (symmetrically, recall) at k , written $\text{prec}@k$ and $\text{rec}@k$.

It is easy to see that the larger this k is, the more we trade precision for recall – in the limit $k = m$ we will simply classify everything as a positive instance, achieving a recall of 1 [MRS08].

The trade-off between precision and recall as k is relaxed can be visualized in a precision-recall curve such as those of figure 2.9: figure 2.9a shows a healthy-looking curve, in which a reasonable trade-off between precision and recall can be achieved, as signified by the large area under the curve.

Conversely, figure 2.9b shows an extremely unreliable classifier where good precision can be had only with small values of k and, conversely, good recall can be had only with large values of k which, however, results in decreased precision due to the large number of false positives that are picked up.

Mean Average Precision

Average Precision, or AP, is a synthetic measure of the classifier's performance on a single prediction across all choices of k where k is a relevant label.

Let first Average Precision be defined as follows⁵:

⁵ [LUB⁺16] gives the equivalent definition

$$\text{iAP}(x) = \frac{1}{R} \sum_{j=1}^m \frac{r_j}{j} \delta(x, t_j)$$

Where r_j is the number of relevant predictions (true positives) for the top scoring j results, R is the number of relevant labels and $\delta(x, t_j) = 1$ iff label t_j is relevant to x .

Therefore,

$$\text{iAP}(x) = \frac{1}{R} \left(\sum_{k \in \{1 \leq k \leq m: \delta(k,x)=1\}} \frac{r_j}{j} \right) + \left(\sum_{k \in \{1 \leq k \leq m: \delta(k,x)=0\}} 0 \right) = \frac{1}{R} \sum_{k \in \{1 \leq k \leq m: \delta(k,x)=1\}} \frac{r_j}{j}$$

$$\text{AP}(\mathbf{p}) = \frac{1}{|\mathbf{R}|} \sum_{k \in \mathbf{R}} (\text{prec}@k(\mathbf{p}))$$

where again $\mathbf{p} = \langle \hat{y}, y \rangle$ and $\mathbf{R} = \{k : y_{(k)} = 1, \text{ i.e. } k\text{-th label is relevant}\}$.

This is graphically depicted in figure 2.12.

Let then S be a set of pairs \mathbf{p} ; then Mean Average Precision is:

$$\text{mAP}(S) = \frac{1}{|S|} \sum_{\mathbf{p} \in S} \text{AP}(\mathbf{p})$$

mAP is a particularly useful metric in the absence of information about the cost or risk of misclassifications, when it is necessary to evaluate the trade-off between different types of classification error [EVGW⁺10].

Per-image and per-label metrics

Recall that we have m labels and some number n of training instances in our dataset – intuitively, with two dimensions there are two ways to define and compute precision and recall, depending on whether $\mathbf{p} = \langle y, \hat{y} \rangle$ is

- the pair of the vector of predicted labels and ground truth for an image instance or
- the pair of predicted images carrying a given label and ground truth

Suppose instead $P = \langle Y, \hat{Y} \rangle$ is a pair of $n \times m$ matrices representing ground truth and predictions, where $Y_{i,j} = 1$ if label j is relevant to image i and symmetrically \hat{Y} .

We define the mean per-image precision prec_{img} (resp. the mean recall rec_{img}) as the performance achieved by the classifier in the former case:

$$\text{prec}_{\text{img}}(P) = \frac{1}{m} \sum_{j=1}^m \frac{\text{TP}(\text{col}_j(P))}{\text{TP}(\text{col}_j(P)) + \text{FP}(\text{col}_j(P))} \in \mathbb{R}^n$$

$$\text{rec}_{\text{img}}(P) = \frac{1}{m} \sum_{j=1}^m \frac{\text{TP}(\text{col}_j(P))}{\text{TP}(\text{col}_j(P)) + \text{FN}(\text{col}_j(P))} \in \mathbb{R}^n$$

This is depicted graphically in figure 2.10.

Symmetrically, per-label precision:

$$\text{preclabel}(P) = \frac{1}{n} \sum_{i=1}^n \frac{\text{TP}(\text{row}_i(P))}{\text{TP}(\text{row}_i(P)) + \text{FP}(\text{row}_i(P))} \in \mathbb{R}^m$$

$$\text{rec}_{\text{label}}(P) = \frac{1}{n} \sum_{i=1}^n \frac{\text{TP}(\text{row}_i(P))}{\text{TP}(\text{row}_i(P)) + \text{FN}(\text{row}_i(P))} \in \mathbb{R}^m$$

Where \mathbf{R} is precisely $\{1 \leq k \leq m : \delta(k, \mathbf{x}) = 1\}$ and $\frac{r_j}{j}$ equals $\frac{\text{TP}}{\text{TP} + \text{FP}}$ if *exactly* the first j predictions are considered, that is $\text{prec}@j$.

Do note that average precision can be “cheated” by choosing an appropriate arbitrary ordering for the labels, but only on a single example. The advantage would cancel out when computing mean average precision, provided the dataset is balanced.

The per-image and per-label $\text{prec}_{\text{img}@k}$, $\text{prec}_{\text{label}@k}$, $\text{rec}_{\text{img}@k}$, $\text{rec}_{\text{label}@k}$ are defined similarly.

Note that the per-image metrics on P are the per-label metrics on P^T .

Similarly, we can also compute per-image and per-label mean average precision [LUB⁺16]:

$$\text{mAP}_{\text{img}}(P) = \frac{1}{n} \sum_{i=1}^n \frac{1}{|J_i|} \underbrace{\sum_{k \in J_i} \text{prec}_{\text{img}@k}(\text{row}_i(P))}_{\text{AP}_{\text{img}}(i)}$$

where $J_i = \{j : j\text{-th label is relevant for } i\text{-th image}\}$, and symmetrically

$$\text{mAP}_{\text{label}}(P) = \frac{1}{m} \sum_{j=1}^m \frac{1}{|I_j|} \underbrace{\sum_{k \in I_j} \text{prec}_{\text{label}@k}(\text{col}_j(P))}_{\text{AP}_{\text{label}}(j)}$$

Since mAP_{img} is averaged over images, each test image contributes equally to mAP_{img} , as opposed to $\text{mAP}_{\text{label}}$, where each label contributes equally.

mAP_{img} is biased toward frequent labels, while $\text{mAP}_{\text{label}}$ can be easily affected by the performance of rare labels [LUB⁺16].

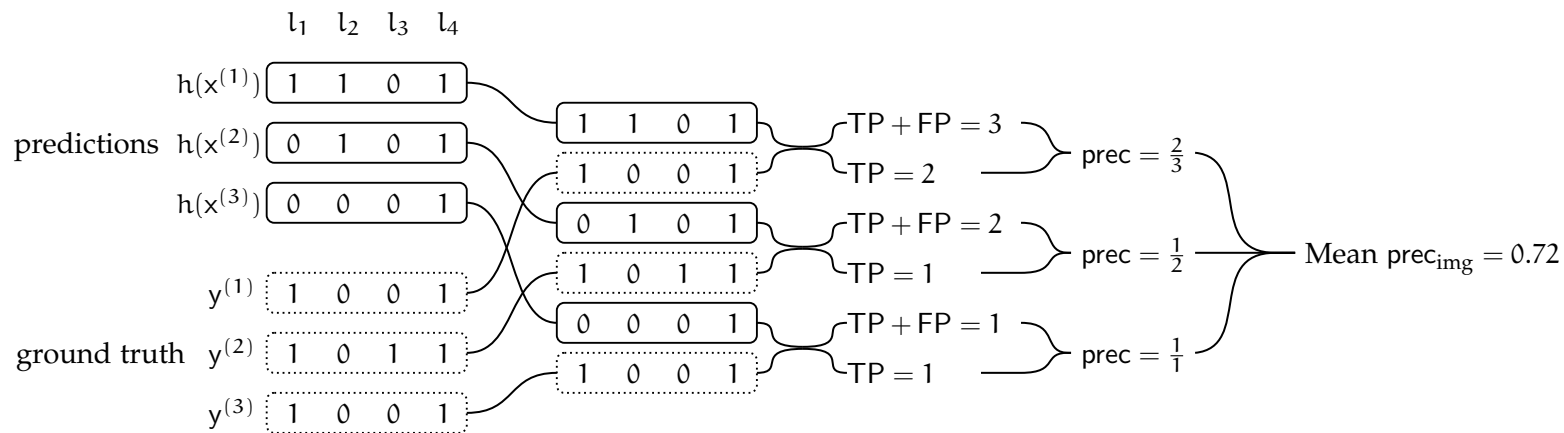


Figure 2.10: Computing per-image mean precision; transpose for per-label mean precision

34

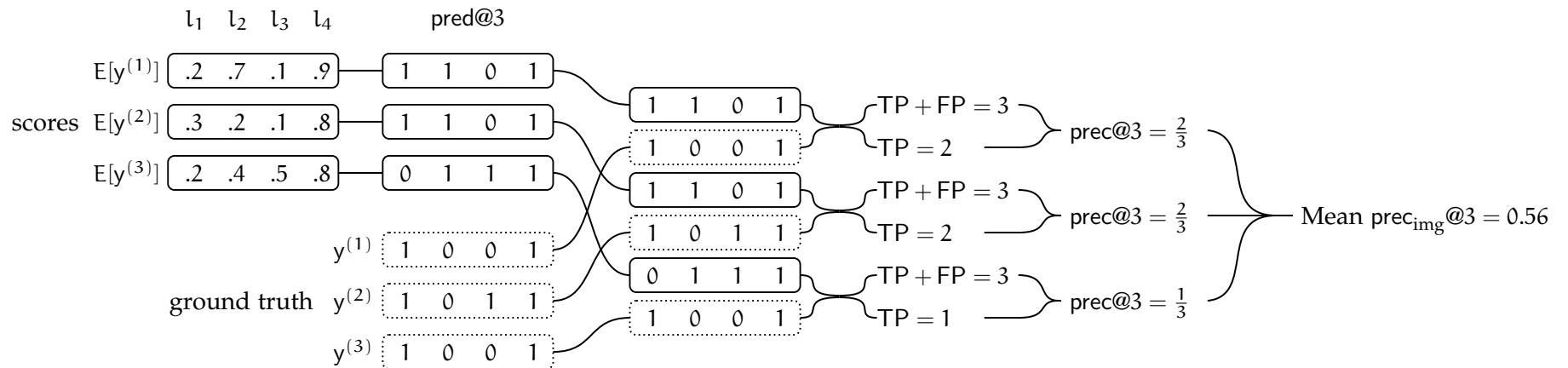


Figure 2.11: Computing per-image mean precision at k; transpose for per-label mean precision at k (note that the prediction vector is *not* sorted by score in the figure)

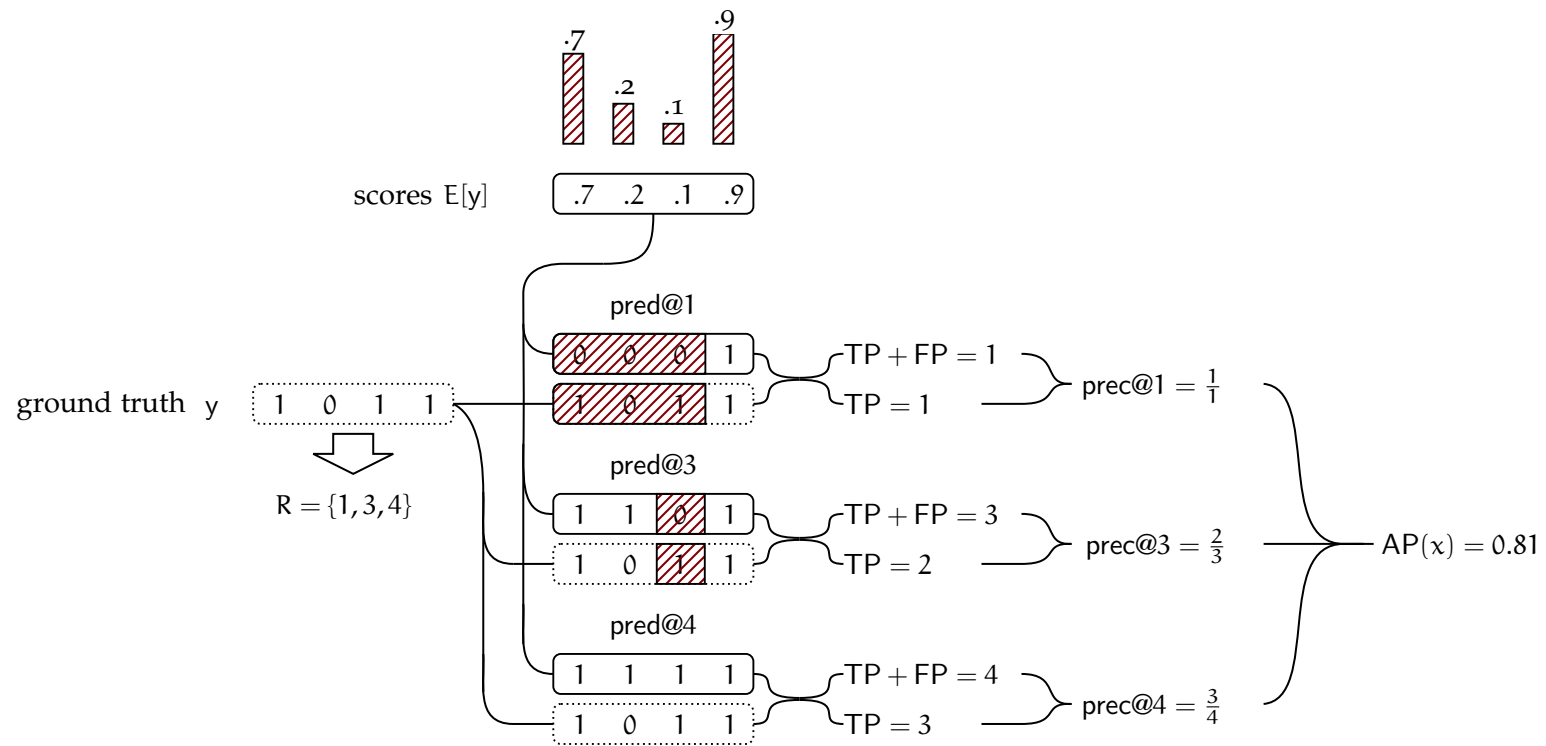


Figure 2.12: Computing AP for an instance (note that the prediction vector is *not* sorted by score in the figure)

2.3.2 CNNs and Deep Learning

Convolutional neural networks are a kind of architecture especially suited for processing data that enjoys from some kind of spatial locality, particularly visual data.

Whereas in a traditional, “fully connected” neural network every unit is connected to all units belonging to the previous layer convolutional networks exhibit *sparse interactions* between units – thus exploiting locality.

Local features are thus extracted and combined to form higher-order features, a technique popular also in the field of classical computer vision [Lec89].

Recall from equation (2.5) and equation (2.6) that “ordinary” multi-layer networks use the dot product operation followed by a possibly non-linear activation function to compute the output of a unit, and, consequently, of a layer; a convolutional neural network is defined thusly:

Definition 10 (Convolutional Neural Network). A neural network that uses the convolution operation instead of the dot product in at least one of its layers [GBC16].

The convolution operation is often accompanied by pooling; networks such as the one represented in figure 2.14, made up of layers of convolution and pooling operations which feed into fully connected subnets one after another are very common ([LBB⁺98], [SZ15]), but are by no means the only possible architecture (a significant counterexample being [HZRS16]; [KSH12] is a slight variation in that it employs two parallel pipelines for ease of implementation).

Convolutional layers act as feature detectors tasked with learning to recognize instances of a particular feature anywhere in the input plane (called “translation invariance”) [LBD⁺89], with somewhat approximate information about its position.

As evidenced by [NHH15], layers tend to operate in a semantically hierarchical fashion: lower layers learn lower-level features (ridges, edges, specific colors), whereas upper layers learn features at a higher semantic level (“wheel”, or “truck”, or even “vehicle”) – this is seen visualized in figure 2.13.

Deep convolutional neural networks (as opposed to early, “shallow” CNNs such as [LBB⁺98]) are a staple of deep learning and, in turn, have made the “deep learning” approach popular in computer vision, thanks to the the outstanding performance from systems such as [KSH12] compared to classical computer vision solutions.

Definition 11 (Deep learning). Generally, “deep learning” refers to machine learning systems that free deep learning practitioners from engineering features by hand, particularly – in contrast with “ordinary” representation learning – by the learning a hierarchy of representations [GBC16].

Convolution The (discrete) convolution operation itself is defined as [FP12]:

$$(I * K)_{i,j} = \sum_m \sum_n I_{m,n} K_{i-m,j-n}$$

where I is the input image and K is a two-dimensional *kernel*.

We shall assume that the sum is over a “large enough” range of m and n that all nonzero values are taken into account [FP12].



Figure 2.13: Top single strongest activations for a given feature map projected down to pixel space using the approach of [NHH15], showing the hierarchical nature of the features in the network. Layer 2 responds to corners and other edge/color conjunctions. Layer 3 has more complex invariances, capturing similar textures, Layer 4 is more class-specific, Layer 5 shows entire objects with significant pose variation. Image reproduced from [NHH15].

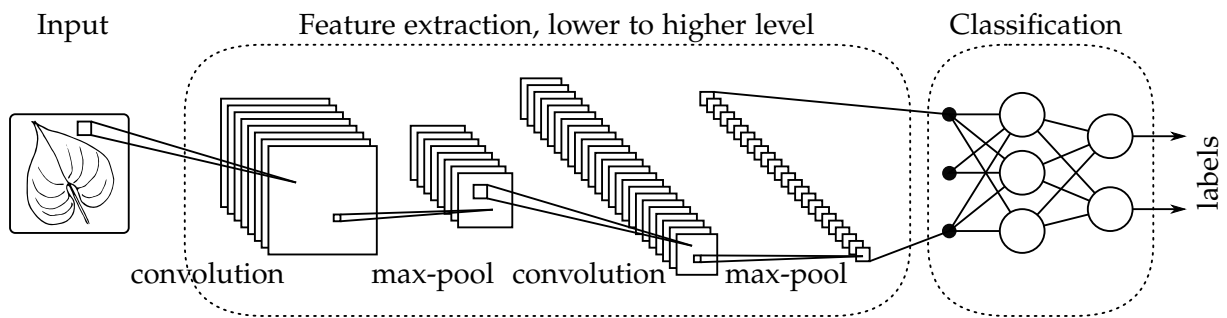


Figure 2.14: Basic structure of a typical CNN

When the kernel is smaller than the input, for every i, j only a subset of neighbouring pixels contribute to $S(i, j)$, thus achieving the locality previously mentioned.

Notice that the same kernel is reused for the whole image, much like a “sliding window” or “moving spotlight”, and, therefore, for the different units connected to the output of the convolution stage; thus affording translation invariance [GBC16].

The kernel K for the convolutional layers are the parameters we’ll seek to learn, along with the weights for the fully connected layers.

Appropriate choices of K can, among other things, result in smoothing, sharpening, edge detection, possibly wrt only one spatial dimension [FP12]; a particularly enlightening interactive demo can be found at [Pow].

Besides the kernel, convolutional layers are characterized by:

1. *depth* – roughly, the number of filters “looking” at each pixel and, therefore, the number of features the layer will learn, and
2. *stride* – the number of pixels by which the filter is moved at every iteration

these are usually fixed, non-learnable parameters.

Pooling Typically, in a convolutional layer, convolution is followed by a nonlinear stage and then by a pooling stage.

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs; its purpose is to progressively reduce the spatial size, and, consequently, to reduce the amount of parameters [cs219] [GBC16].

The benefits are manifold: the amount of computation is made manageable, overfitting is kept under control and invariance to small translations is achieved.

Average pooling is a frequent choice; recently max-pooling has become gained popularity.

Pooling layers are also characterized by field size and stride; these are usually fixed parameters.

An extensive introduction to CNNs is found in [cs219]; a rather nice interactive demonstration of a classic LeNet-style CNN is found at [Har15].

2.3.3 Transfer learning and pretrained models

Training a CNN from randomly initialized weights is expensive in terms of data and time.

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad K = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad I * K = \begin{bmatrix} -13 & -20 & -17 \\ -18 & -24 & -18 \\ 13 & 20 & 17 \end{bmatrix}$$

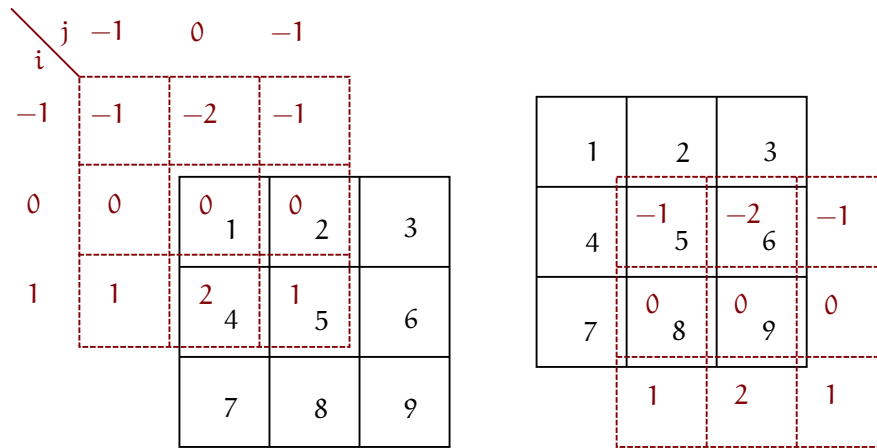
$$(I * K)_{0,0} = \sum_m \sum_n I_{m,n} \cdot K_{0-m,0-n} = 13$$

$$(I * K)_{1,0} = \sum_m \sum_n I_{m,n} \cdot K_{1-m,0-n} = -20$$

⋮

$$(I * K)_{2,2} = \sum_m \sum_n I_{m,n} \cdot K_{2-m,2-n} = 17$$

(a) Input, kernel, result and computation



(b) Visualization of step $(I * K)_{0,0} = \sum_m \sum_n I_{m,n} \cdot K_{0-m,0-n} = 13$

(c) Visualization of step $(I * K)_{2,2} = \sum_m \sum_n I_{m,n} \cdot K_{2-m,2-n} = 17$

Figure 2.15: An example of a convolution operation (adapted from [son19])

$$\text{pool}_{\max(2,2)} \left(\begin{bmatrix} 1 & 1 & 2 & 4 \\ 5 & 6 & 7 & 8 \\ 3 & 2 & 1 & 0 \\ 1 & 2 & 3 & 4 \end{bmatrix} \right) = \begin{bmatrix} 6 & 8 \\ 3 & 4 \end{bmatrix} \quad \text{pool}_{\text{avg}(2,2)} \left(\begin{bmatrix} 1 & 1 & 2 & 4 \\ 5 & 6 & 7 & 8 \\ 3 & 2 & 1 & 0 \\ 1 & 2 & 3 & 4 \end{bmatrix} \right) = \begin{bmatrix} \frac{13}{4} & \frac{21}{4} \\ 2 & 2 \end{bmatrix}$$

(a) Example of max-pooling with field size and stride 2

(b) Example of avg-pooling with field size and stride 2

Figure 2.16: Examples of pooling operations

The hierarchical structure discussed in the previous section and highlighted in figure 2.13 fortunately lends itself well to training the network once on a large dataset for a given task and, at a relatively modest expense in terms of data and time, adapt the model to a new dataset: this is achieved by extracting features from one of the terminal layers of a pre-trained CNN, in which the weights are “frozen”, and using them to feed a new one-layer (or multi-layer) fully connected network, which will be the only one to be trained.

An approach often used in practice is to “dump” fixed features for the layer of interest once, instead of running the whole architecture.

Alternately, is possible, at an additional cost of time and memory, to fine-tune the weights of more than one layer of the pretrained network: note that the representational power of a large network might be underutilized otherwise – in the case of ImageNet for example, which contains many dog breeds, a significant portion of the representational power of the ConvNet may be devoted to features that are specific to differentiating between dog breeds [cs219], which might not be of interest.

ImageNet [DDS⁺09], which contains 1.2 million images with 1000 categories, is popular as a dataset [cs219] and several pre-trained models are available for download.

2.4 Distributional word representations

In the field of natural language processing, one-hot encoding has enjoyed continued popularity as a way to represent individual words, particularly in combination with the “bag of words” model, which amounts to discarding order and grammar, in favour of simply keeping count of the words that appear in a document.

Definition 12 (One-hot). Given a dictionary of τ words $\langle w_1, w_2 \dots w_\tau \rangle$ the one-hot vector for the i -th word is:

$$\text{onehot}(w_i) \triangleq e_i^\tau$$

where e_i is the vector of size τ which is null except for its i -th entry.

Definition 13 (Bag of words representation). Given a document consisting of a sequence of words $D = \langle w_{i_1}, w_{i_2} \dots w_{i_n} \rangle$, and a representation $\pi(w)$ (e.g. $\text{onehot}(w)$), the document’s bag-of-words representation $\text{bow}(D)$ amounts to

$$\text{bow}(D) \equiv \sum_{i \in \{i_1, i_2, \dots, i_n\}} \pi(w_i)$$

If $\pi(w) = \text{onehot}(w)$, this results in a vector $[v_1 \ v_2 \ \dots \ v_\tau]$, where $v_i = k$ iff the i -th word in the dictionary appears k times in a text.

This comes with the following limitations:

1. Results in an high dimensional, sparse vector
2. No information about the word order is preserved, by design
3. No explicit information can be had about word *similarity* - both syntactically and in terms of meaning⁶

⁶The two are not necessarily disjoint: e.g. in a large number of languages words with a root in common often have a semantic relationship

The last point is of particular interest to us.

For example, suppose $w_i = \text{"duck"}$, $w_j = \text{"chicken"}$, $w_k = \text{"UFO"}$.

Clearly the bag-of-words encoding of

I'll have the roast chicken, please.

will differ from that of

I'll have the roast goose, please.

exactly as much as it differs from that of

I'll have the roast UFO, please.

Some information is therefore lost.

This, along with the first point, is addressed by *distributed representations* [Hin86], also called distributional representations or *word embeddings*, in which every word is encoded by a pattern distributed over a smaller number of dimensions, in such a way that, while the individual dimensions become meaningless, the similarity between vectors can capture the similarity between words.

In [MCCD13], the authors propose techniques to *learn* such representations automatically, optimizing the resulting vector space to capture word similarity in human judgment as approximated by the frequency of contextual co-occurrence of words.

Precomputed dictionaries of so-called word2vec embeddings, obtained with such techniques exposed in the above-mentioned paper, are available.

WordNet [Mil95] is a database of English words, curated by human experts, that represents words and the relations that exist between them, such as synonymy, hypernymy (the relation between a word and a "more general one"), meronymy (the relation of "being a part of", variously defined).

In [SBRS], the authors attempt to use WordNet to construct word embeddings that capture the relationship between words as represented and recorded in the WordNet database, thus leveraging and incorporating the semantic knowledge contained therein.

This page intentionally left blank.

Chapter 3

Literature review

As seen in [LUB⁺16], there is a growing literature on the problem of image auto-annotation and the more general problem of predicting relevance of a tag to an image using metadata, which complements the large body of literature that focuses on exclusively visual approaches [SWS⁺01].

3.1 Metadata-based models

[GVS10] considers a scenario in which only visual data is available at test time, but metadata from social media websites are available at training time and can be leveraged to improve the visual-only classifier using semi-supervised learning.

They assume having a set \mathcal{L} of training images with ground truth labels and metadata, plus a set of training images \mathcal{U} with metadata but no ground truth.

The authors envision a two-step process to leverage the metadata; in principle:

- Firstly, they use multiple kernel learning to learn a joint visual-textual classifier f_c from \mathcal{L} , and use it to estimate the class labels for the images in \mathcal{U} , which they assume are (substantially) accurate.
- Then, they train a visual-only SVM classifier from all training examples in $\mathcal{L} \cup \mathcal{U}$ (where the ones from \mathcal{U} have synthetic labels from f_c).

In [LXH⁺17], the authors consider that the common pattern of feeding the output of a CNN into an RNN via a visual embedding output by the CNN imposes upon the RNN the duty of predicting the visual concepts *and* modelling their correlations for generating structured annotation output, with a detrimental effect on the end-to-end training of the CNN and RNN; they suggest using a semantically regularised embedding layer to decouple the CNN and RNN.

3.2 Neighbour-based models

Several authors have proposed models that use metadata to construct neighbourhoods from which to draw information.

In [GMVS09], the authors present a nearest neighbour model that predicts the presence or absence of tags in an annotated image by taking a weighted combination of the tag

absence/presence among neighbours, under the assumption that a distance measure is provided – possibly visual similarity.

The authors attempt to predict the presence or absence of a tag w in image i ($y_{iw} \in \{-1, +1\}$) as a weighted sum over the training images, weighted by a certain weight π_{ij} , measuring the weight of image j wrt i :

$$p(y_{iw} = +1) = \sum_j \pi_{ij} p(y_{iw} = +1|j)$$

where

$$p(y_{iw} = +1|j) = \begin{cases} 1 - \varepsilon & \text{iff } y_{jw} = +1 \\ \varepsilon & \text{otherwise} \end{cases}$$

to avoid zero prediction probabilities.

The weight π_{ij} , defined as 0 for $i = j$, can be obtained from NN rank or directly as distance measure.

The authors stress that the dependencies between keywords in the training data are not explicitly modeled, but are implicitly exploited.

[GHF12] propose deriving a measure of similarity from Flickr image groups which can then be used to predict how unseen examples belong to Flickr groups; a key observation there is that Flickr groups (and, we speculate, Flickr tags) can embed more complex implicit knowledge about image similarity than the “usual” hand-applied tags and labels from datasets do.

The authors exemplify this by noting the existence of a Flickr group called “No-Flash Night Shots”.

[ML12] observe that social-network metadata is relational rather than categorical, and can be used to explicitly construct neighbourhoods, thus providing reliably provide context not contained in the image itself.

The model proposed by the authors is inherently multimodal – i.e. different sorts of metadata are taken into account to determine neighbours (figure 3.1).

However, the approach used implies two limitations:

- Firstly, it is not possible to make predictions for a single image; the per-class predictor $\hat{Y}_c(\chi, \Theta_c)$ proposed by the authors, in which χ s.t. $|\chi| = N$ is a dataset and Θ_c is a parameter vector for category c , has codomain $\{0, 1\}^N$ and assigns a 0 or a 1 in i -th position iff image i is predicted to belong to category c , and has the form

$$\hat{Y}_c(\chi, \Theta_c) \triangleq \arg \max_{Y \in \{-1, 1\}^N} \sum_{i=1}^N y_i \cdot \langle \phi_c(x_i), \theta_c^{\text{node}} \rangle + \sum_{i,j=1}^N \sum_{j=1}^N \delta(y_i = y_j) \langle \phi_c(x_i, x_j), \theta_c^{\text{edge}} \rangle$$

where $\phi_c(x_i, x_j)$ is a feature vector encoding the relationship between x_i and x_j and $\delta(y_i = y_j)$ iff the predictions for y_i equal those for y_j .

- The classifier learning is *parametric*: the optimization process finds optimal Θ_c wrt a given χ and its associated ground truth Y_c . It is not robust to changes in the data-generating distribution or to the addition of a class, which require retraining.

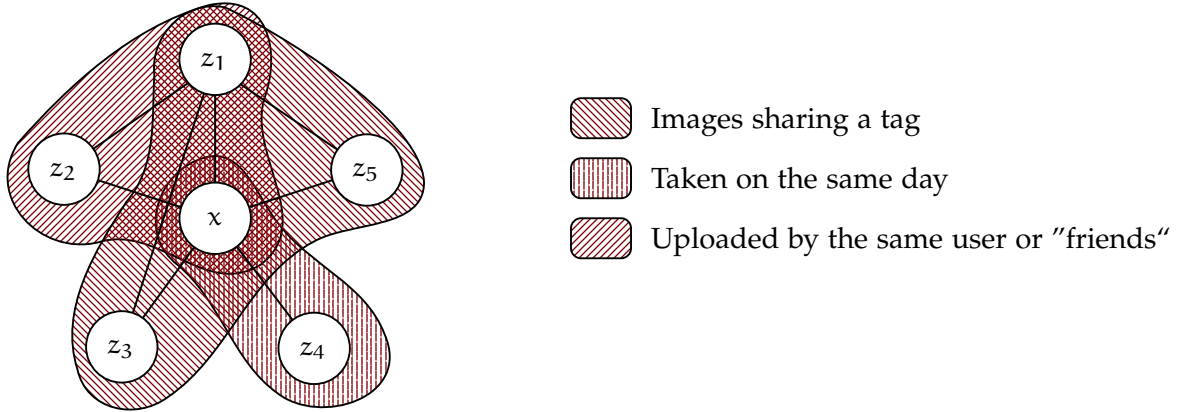


Figure 3.1: The inherently multimodal relational model proposed in [ML12]; actual distance not pictured

This makes the proposed model’s usefulness limited for popular applications such as social media websites or on-line, changing databases; moreover, it does not use any visual information.

In [JBFF15], the authors try to build upon the same intuition while avoiding the previously-mentioned disadvantages.

Social network metadata is used to generate neighbourhoods of images nonparametrically, then the proposed model operates on these neighbourhoods with a parametric model that learns to optimally leverage the visual features of neighbouring images.

Central to the model is the notion of generating and sampling from *candidate neighbourhoods*. Let $D = \{(x, y) | x \in X, y \subseteq Y\}$ be a dataset associating each image x with a set y of labels.

A neighbourhood is an element of $Z = 2^X$; for each image x a set of candidate neighbourhoods $Z_x \subseteq Z$ is generated.

At training time Z_x is (non-uniformly) drawn from training images, whereas at test time it is drawn from the test set using only information contained in the test dataset, thus achieving non-parametricity wrt neighbourhoods.

A function $f(x, z; w)$ parametrized over learned w is used at test time to predict label scores for x conditional to z being a neighbour.

The predicted label scores conditional to Z_x being a neighbourhood are given by

$$s(x; w) \triangleq \frac{1}{|Z_x|} \sum_{z \in Z_x} f(x, z; w)$$

Neighbourhoods are generated using a nearest-neighbour approach: each type of metadata has a vocabulary T of possible values and associates each image $x \in X$ with a subset $t_x \subseteq T$; the Jaccard distance¹ is used to compute nearest neighbours, defined as:

$$J(x, x') \triangleq 1 - \frac{|t_x \cap t_{x'}|}{|t_x \cup t_{x'}|} \quad (3.1)$$

With $J(x, x) \triangleq 0$.

¹The complement of Jaccard similarity, also known as intersection-over-union

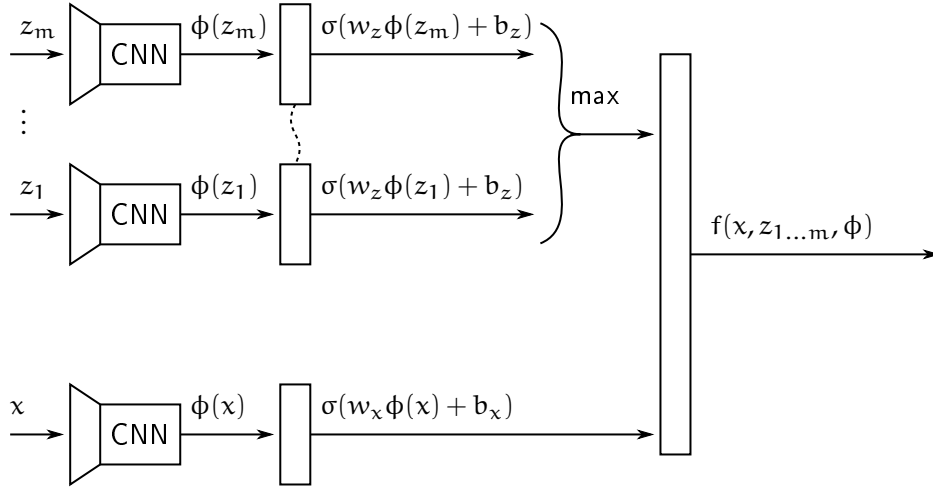


Figure 3.2: Neural architecture from [JBFF15]; x is the image to be classified; $z_1 \dots z_n$ are its neighbours. Dashed line signifies weight sharing.

Labels are then predicted with the parametric architecture summarized in figure 3.2; notice how the weights for neighbours are shared.

In the end the learned classifier is

$$f(x, \theta; z) = \mathbf{w}_y \begin{bmatrix} v_x \\ v_z \end{bmatrix} + b_y$$

where z is a vector of neighbours obtained nonparametrically, x is the image to be classified, θ are learnable parameters $\mathbf{w}_x, \mathbf{w}_z, \mathbf{w}_y, b_x, b_z, b_y$ and

$$v_x = \sigma(\mathbf{w}_x \phi(x) + b_x)$$

$$v_z = \max_{i=1 \dots m} (\sigma(\mathbf{w}_z \phi(z_i) + b_z))$$

where $\phi(x)$ are visual features extracted for x .

3.3 Semantic models

[HZD⁺16] starts from the explicit observation observes that diverse levels of visual categorization are possible; depending on the level of abstraction desired, different labels could be just as “right” – e.g. “pug” vs. “dog” vs. “animal” – we observe that this is connected to the notion of semantic gap (see chapter 1) more than to the notion of sensory gap.

The authors propose modeling categorization of visual concept at different levels, using a graph-like structure that encodes both hierarchical and horizontal relationships, including negative correlation, and using a stacked neural network to make predictions.

The authors use external knowledge of the ontological relationship between labels from a WordNet taxonomy [Mil95].

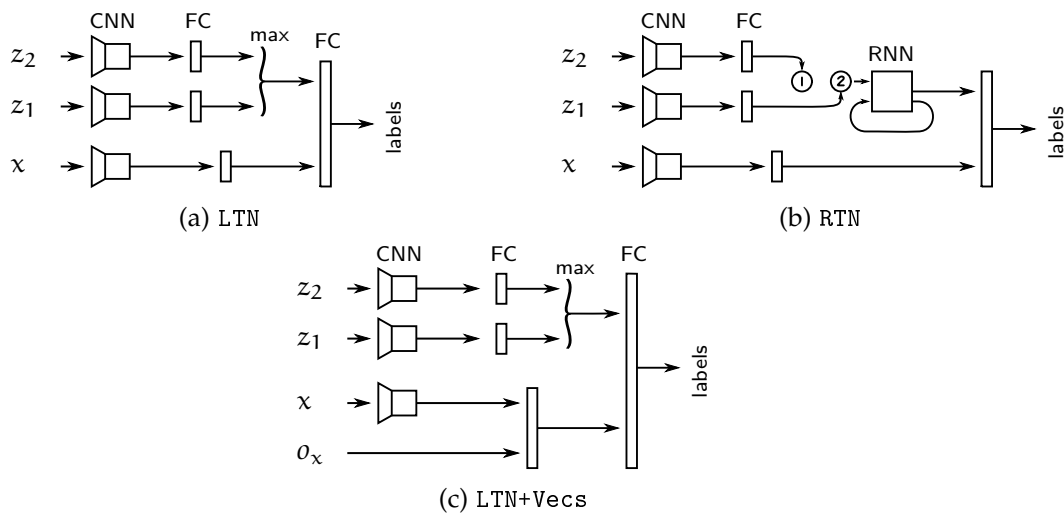


Figure 4.1: Visual neural architectures

Chapter 4

Our models

We describe, explain the rationale and examine the performance of a set of models that build upon [JBFF15] in different ways.

Two main families of models that leverage metadata neighbourhoods and are built around a neural architecture are explored: visual and joint models.

Both arise out of – or, more accurately, are “frameworks” on which to instantiate – a combination of parameters: the specific neural architecture, which we explore in section 4.3 the semantic transformations used on metadata and the distance measures used to compute neighbourhoods, which we explore in section 4.2, and finally neighbourhood size, which we explore in section 4.1.

4.1 Neighbourhood generation and size

What all models have in common is that, exactly as in [JBFF15], discussed in chapter 3 and on which this thesis builds upon:

1. First, a neighbourhood Z_x of images, where x is the image to be classified, is generated

using metadata

2. Then, the network is trained to classify x given the image x and its neighbours in Z_x

Neighbourhood generation is parametrized over neighbourhood size m and max rank M in the following way:

Let Z_x be the M -nearest neighbours of x according to a distance measure δ .

The set of *candidate* neighbourhoods for an image x is the set

$$Z_x \triangleq \{s \in \mathcal{P}(Z_x) : |s| = m\}$$

The prediction $s(x, \theta)$ is the average

$$s(x, \theta) = \frac{1}{|Z_x|} \sum_{z \in Z_x} f(x, \vec{z}; \theta)$$

of $f(x, \vec{z}; \theta)$ over all candidate neighbourhoods, where x is the image to be classified, $\vec{z} = \langle z_1, z_2, \dots, z_m \rangle$ are the images in the the neighbourhood and $f(x, \vec{z}; \theta)$ is the output of a given neural architecture described in section 4.3.

The model is trained by minimizing:

$$\theta^* = \arg \min_{\theta} \sum_{\langle x, y \rangle \in \mathcal{D}_{\text{train}}} \mathcal{L}(s(x, \theta), y)$$

This general framework, which is exactly that of [JBFF15], is depicted in figure 4.4.

Note that neighbours are ordered according to their distance when fed to the neural network – this is a detail not mentioned nor relevant in [JBFF15], since the neural model found therein cannot discriminate between its inputs, due to the effect of weight sharing and of the max-pooling operation.

A variation is had in the case of joint models, in which the metadata is fed directly to the neural network, possibly after a transformation step π , as discussed in section 4.2, which in our proposed models will involve a lookup in a dictionary of semantic embeddings.

In this case, the prediction $s(x, \theta)$ is the average of

$$f(x, \pi(o_x), \vec{z}, \pi(\vec{o}_z); \theta)$$

over all candidate neighbourhoods, where x is the image to be classified, o_x is the metadata vector for image x and $\pi(o_x)$ is its transform (possibly $\pi = \text{id}$) and $\vec{z} = \langle z_1, z_2, \dots, z_m \rangle$ are the images in the the neighbourhood.

We shall use $\pi(\vec{o}_z)$ as shorthand for $\text{map}(\pi, \vec{o}_z) \triangleq \langle \pi(o_{z_1}), \pi(o_{z_2}), \dots, \pi(o_{z_m}) \rangle$, where \vec{o}_z are metadata vectors for the neighbourhood.

The resulting framework is depicted in figure 4.5.

To assist intuition, one example of an image from the dataset NUS-Wide and its 6 nearest metadata neighbours wrt Jaccard distance is reproduced in figure 4.3.

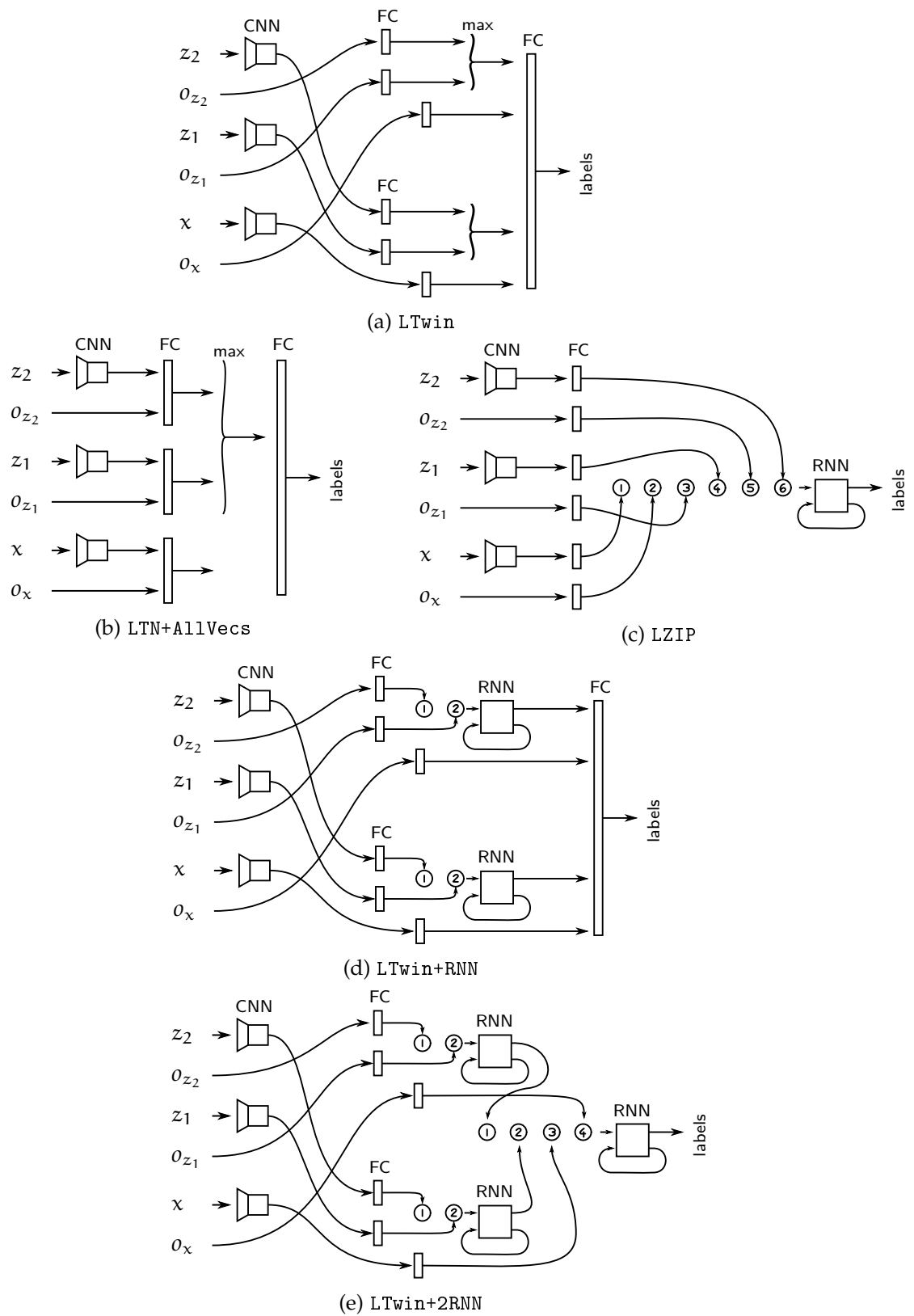


Figure 4.2: Joint neural architectures








Image: 163792		grass	centipede, yellow, naturesfinest, k1ood, macro, pentax, kit, eyes, animals, grass, chenille, nature, 1855, johannpix, caterpillar
Neighbour: 140470		animal	flickrdiamond, animalkingdomelite, dragonfly, naturesfinest*, k1ood*, macro*, pentax*, wild, kit*, animals*, blue, damselfly, green, nature*, blueribbonwinner, 1855*, diamondclassphotographer, closeup, johannpix*, libellule
Neighbour: 140175		sun, sky, flowers, clouds	yellow*, naturesfinest*, k1ood*, pentax*, flash, soe, kit*, outdoors, overtheshot, 1855*, colors, sun, flowers, johannpix*, sky, tulips, fillin
Neighbour: 15106		animal	yellow*, macro*, 5hits, selectivecolorization, animals*, selectivecolor, bird, nature*, chicken, chick, beak, baby, bw
Neighbour: 114315		leaf, flowers, plants	yellow*, naturesfinest*, yellowflower, macro*, greatflowersmacro, pottedplant, greenhouse, nature*, olympus, ilovenature, datura, flower, botany, diamondclassphotographer, wintero8, goldenmix, brugmansia, trumpetflower
Neighbour: 86282		animal	elephants, wildlife, southafricanwildlife, naturesfinest*, borntobewild, goodmanandy, animals*, goodman, btbw, african, nature*, wildlifesouthafrica, wildlifeinsouthernafrica, wildilfephotographer, andygoodman, africanwildlifephotographer, southernafricanwildlife
Neighbour: 140304		rocks, mountain, water	wood, alps, k1ood*, pentax*, water, exposure, waterfall, kit*, long, 1855*, johannpix*, queyras, mountain, france

Figure 4.3: An image and some neighbours, sorted nearest to farthest. Tags marked with * are shared with the image by the neighbours. Note the decreasing semantic and visual relevance. Note also the relative arbitrariness of the ground truth label(s): there does not seem to be a particular reason why image 163792 should be “grass” instead of “animal”

NN search in Database

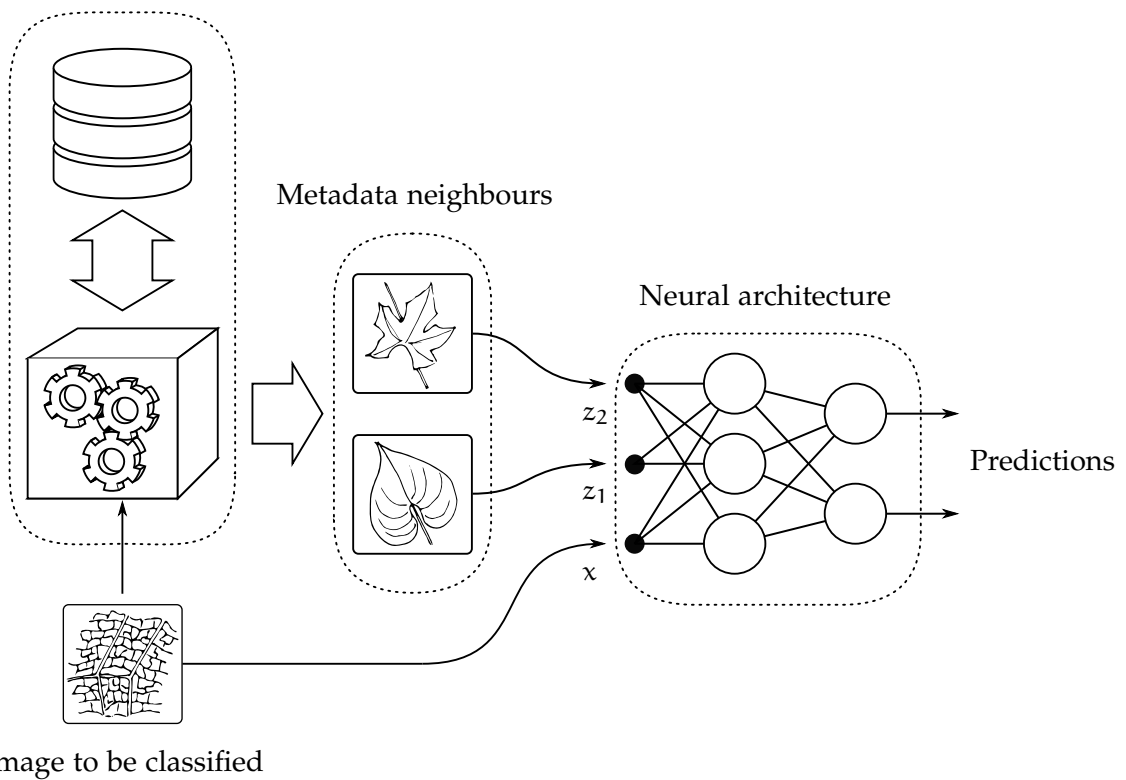


Figure 4.4: General framework for visual models. Note that the image to be classified is already in database.

NN search in Database

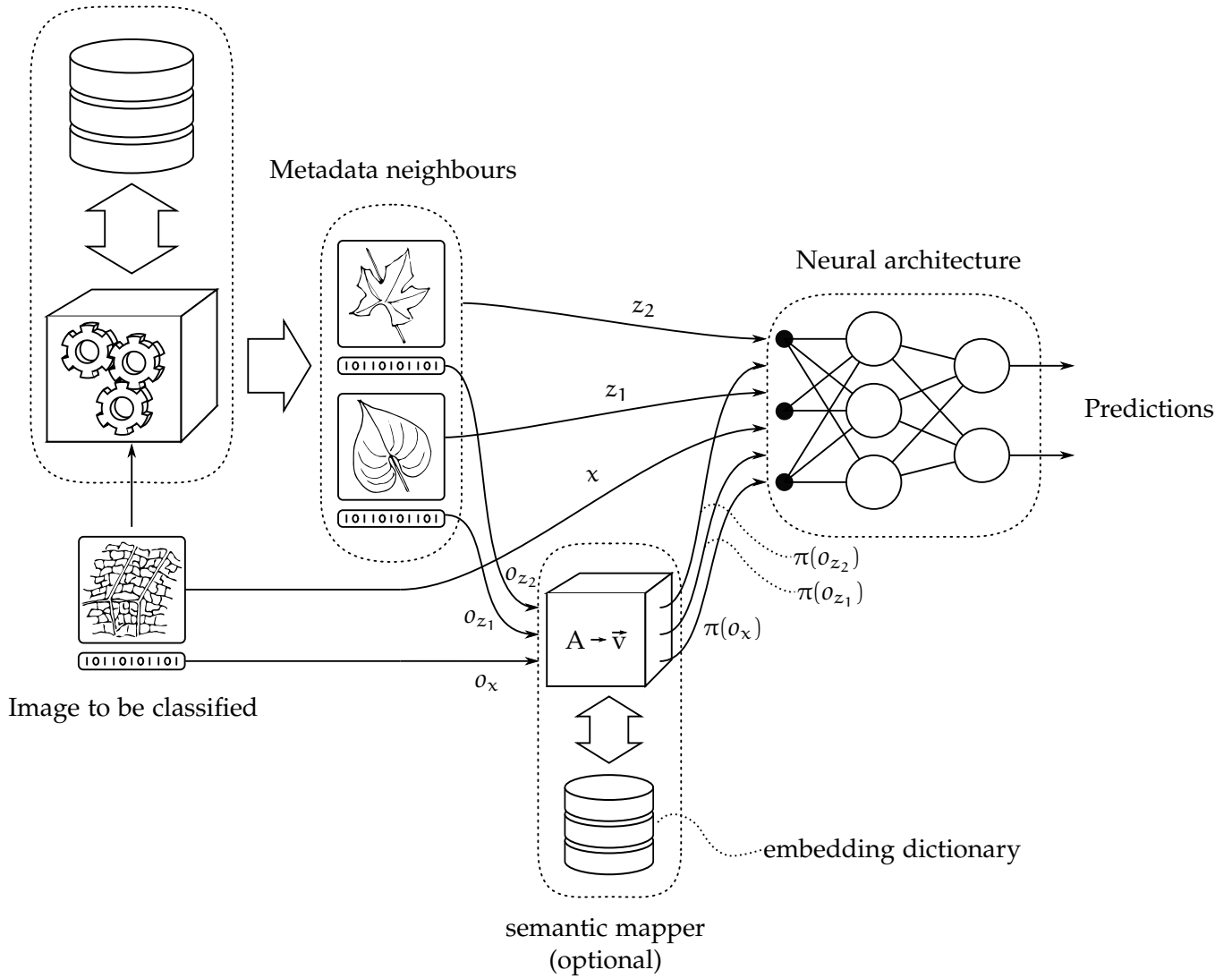


Figure 4.5: General framework for joint models; optional use of semantic transformations in neighbourhood generation not pictured. Note that the image to be classified is already in database.

4.2 Metadata semantic mappings π and distance measures δ

In our models, metadata is both used to generate neighbourhoods *and* is supplied to those joint network architectures that explicitly take metadata as input(s), as we will see in section 4.3.

In [JBFF15] social network “tags”¹ are identified as the type of metadata for image classification that yields the largest performance improvement over a visual-only classifier.

We will focus on this sort of metadata, which we will assume is represented as binary vectors in its starting representation:

Definition 14. Let x be an image and $t_{(1)}, t_{(2)}, \dots, t_{(n)}$ be all tags relevant for x chosen from a vocabulary, i.e. a set TAGS of τ tags.

The binary vector $o_x \in \{0, 1\}^\tau$ for the image is then the sum of the one-hot vectors (section 2.4) for each of its tags:

$$o_x \triangleq \sum_{i \text{ s.t. } t_{(i)} \in \{t_{(1)}, t_{(2)}, \dots, t_{(n)}\}} e_i^\tau$$

We will consider metadata transforms that map a vector o_x to a *semantic space*:

$$\pi : \{0, 1\}^\tau \rightarrow \mathbb{R}^n$$

Having one such transformation, we can then perform one of the following, both, or none (which amounts to using $\pi = \text{id}$):

1. Use the transformed metadata, along with a suitable distance measure δ , for generating the neighbourhood Z_x for x
2. Use the transformed metadata $\pi(o_x)$ as input to joint network architectures that take metadata input

Recall that one of the key advantages of the framework in [JBFF15] is its nonparametricity wrt metadata and, therefore, its adaptability to different metadata at test time.

However, it is worth noting at this point that not all joint models are nonparametric in this sense, and that π has a key role in enabling nonparametricity.

It is clear that, unlike visual models, where metadata is used *implicitly*, a neural network trained to make predictions as a function of one or more binary vector becomes useless the very moment the meaning encoded in those vectors changes – which is, when the vocabulary changes.

Moreover, the learned weights are not even of the right *shape* to be applied to binary vectors of different lengths!

This limits the transferability of the learned model to a different database, which is a key concern for us.

Suppose, for example, that the model is trained to predict labels from visual features and binary vectors of the form $\langle t_1, t_2, t_3 \rangle$ where $t_i = 1$ iff the tags “coffee”, “bicycle”, “bear”, respectively, are relevant for a given image.

¹Not to be confused with *labels* we wish to predict

If t_1, t_2, t_3 came to mean, respectively, “coffee”, “hydrant”, “lamp” the model would no longer work well.

Semantic maps π can decouple the low-level bit representation from the semantic meaning, making models learned on a tag vocabulary applicable to a different one, as long as an appropriate π' is available that maps the “new” binary vectors onto the “old” semantic space.

We will discuss choices and significance of π and δ in section 5.1.2.

4.3 Neural architectures

We present a set of neural architectures partitioned in two broad subsets, depicted in figure 4.1 and figure 4.2: visual and joint models.

Visual models, which we discuss first, exclusively take visual images as an input, whereas joint models are directly fed with metadata instead of leveraging metadata *only implicitly*, through neighbourhoods generated from metadata.

4.3.1 Visual architectures

Visual only

This architecture acts as baseline; it simply amounts to a fully connected layer over visual features $\phi(x)$ output by a CNN for an image x .

We have, therefore,

$$f(x, \vec{z}; \theta) = W_y \phi(x) b_y$$

with

$$\theta = \{w, b\}$$

Note that \vec{z} is not used.

LTN

This architecture is depicted in figure 4.1a and is exactly the architecture used in [JBFF15]; we will consider it a “baseline” of sorts as well.

Its output is

$$f(x, \theta; \vec{z}) = w_y \begin{bmatrix} v_x \\ v_z \end{bmatrix} + b_y$$

parametrized by $\theta = \{w_y, b_y, w_z, b_z, w_x, b_x\}$, where $\vec{z} = \langle z_1, z_2 \dots z_m \rangle$ is a vector of neighbours obtained nonparametrically, x is the image to be classified, and

$$v_x \equiv \sigma(w_x \phi(x) + b_x)$$

$$v_z \equiv \max_{i=1 \dots m} (\sigma(w_z \phi(z_i) + b_z))$$

where σ is a ReLU layer.

Note that the weights w_z, b_z are shared among all $z_{1 \dots m}$; the hidden layers v_z and v_x have the same width, which we will denote as h .

RTN

This architecture extends LTN by replacing the max-pooling operation with an RNN, with the expectation that individual neighbours and that, being the RNN strictly more powerful, it can be trained to be more discriminating.

Note that weight sharing among visual pipelines for neighbours is retained. This architecture is depicted in figure 4.1b.

We have

$$f(x, \theta; z) = \mathbf{w}_y \begin{bmatrix} v_x \\ v_z \end{bmatrix} + \mathbf{b}_y$$

parametrized by

$$\theta = \{\mathbf{w}_y, \mathbf{b}_y, \mathbf{w}_z, \mathbf{b}_z, \mathbf{w}_x, \mathbf{b}_x\} \cup \{\mathbf{w}_{\text{RNN}}\}$$

where again x and z are the image to be classified and its neighbours, and

$$v_x \equiv \sigma(\mathbf{w}_x \phi(x) + \mathbf{b}_x)$$

$$v_z \equiv \text{loop}(\text{RNN}(\mathbf{w}_{\text{RNN}}), \langle z_1, \dots, z_n \rangle)$$

Where

$$\text{loop}(\text{RNN}(\mathbf{w}_{\text{RNN}}), \text{seq}) \equiv \text{fst}(\text{fold}(\text{RNN}(\mathbf{w}_{\text{RNN}}), \langle \vec{0}, \vec{0} \rangle, \text{seq}))$$

with²

$$\text{fold}(f, z, \langle x_1, \dots, x_n \rangle) \triangleq \text{fold}(f, f(z, x_1), \langle x_2 \dots x_n \rangle)$$

$$\text{fold}(f, z, \langle \rangle) \triangleq z$$

$$\text{fst}\langle x_l, x_r \rangle \triangleq x_l$$

where $\text{RNN}(\mathbf{w}_{\text{RNN}})(\langle h_{i-1}, o_{i-1} \rangle, x_i) = \langle h_i, o_i \rangle$ is the internal state and output of the RNN given internal state h_{i-1} and input x_i and kernel \mathbf{w}_{RNN} .

In this case, RNN is an LSTM with linear activation: empirically, it has proved itself slightly more performant than a more conventional tanh activation followed by a fully connected layer.

This architecture is depicted in figure 4.1b.

Again, σ is ReLU and the layers v_z and v_x have the same width, which we will denote as h .

4.3.2 Joint architectures

LTN+Vecs

This architecture makes use of the metadata o_x for the image to be classified as input to the same fully connected layer that follows the CNN for the image input x .

It is depicted in figure 4.1c.

²fst, fold are the familiar functional programming definitions as found in e.g. Haskell

Its output is

$$f(x, \pi(o_x), \vec{z}, \pi(\vec{o}_z); \theta) = \mathbf{w}_y \begin{bmatrix} v_x \\ v_z \end{bmatrix} + \mathbf{b}_y$$

Where

$$v_x \equiv \sigma \left(\mathbf{w}_x \begin{bmatrix} \phi(x) \\ \pi(o_x) \end{bmatrix} + b_x \right)$$

and, as in LTN, z is a vector of visual features for neighbours obtained nonparametrically, x is the image to be classified, and

$$v_z \equiv \max_{i=1 \dots m} (\sigma(\mathbf{w}_z \phi(z_i) + b_z))$$

Note that neighbour metadata vectors \vec{o}_z are not used at all.

LTN+AllVecs

This architecture, depicted in figure 4.2b, unlike the previous one, uses metadata vectors o_x for the image to be classified *and* for its neighbours \vec{o}_z .

Its output is

$$f(x, \pi(o_x), \vec{z}, \pi(\vec{o}_z); \theta) = \mathbf{w}_y \begin{bmatrix} v_x \\ v_z \end{bmatrix} + \mathbf{b}_y$$

Where

$$v_x \equiv \sigma \left(\mathbf{w}_x \begin{bmatrix} \phi(x) \\ \pi(o_x) \end{bmatrix} + b_x \right)$$

and

$$v_z \equiv \max_{i=1 \dots m} \left(\sigma \left(\mathbf{w}_z \begin{bmatrix} \phi(z_i) \\ \pi(o_{z_i}) \end{bmatrix} + b_z \right) \right)$$

Notice how it therefore essentially amounts to LTN in which visual features are replaced to a concatenation of features and metadata.

In this case, too, σ is ReLU and the hidden layers v_x and v_z have the same width, which we will denote as h .

LTwin

This architecture, depicted in figure 4.2a, is fed metadata o_x for the image to be classified *and* for its neighbours as $o_{z_1 \dots m}$ – in other words, the same inputs as LTN+AllVecs, but processes them in separate pipelines.

The neighbours are blended with a max-pooling layer, so it is not able to discriminate between nearest and farthest neighbours.

Its output is

$$f(x, \pi(o_x), \vec{z}, \pi(\vec{o}_z); \theta) = \mathbf{w}_y \begin{bmatrix} v_x \\ v_z \\ u_x \\ u_z \end{bmatrix} + \mathbf{b}_y$$

parametrized by

$$\theta = \{\mathbf{w}_y, b_y, \mathbf{w}_{o_z}, b_{o_z}, \mathbf{w}_{o_x}, b_{o_x}, \mathbf{w}_z, b_z, \mathbf{w}_x, b_x\}$$

Where, as in LTN,

$$v_x \equiv \sigma(\mathbf{w}_x \phi(x) + b_x)$$

and

$$v_z \equiv \max_{i=1 \dots m} (\sigma(\mathbf{w}_z \phi(z_i) + b_z))$$

and

$$u_x \equiv \sigma(\mathbf{w}_{o_x} \pi(o_x) + b_{o_x})$$

and

$$u_z \equiv \max_{i=1 \dots m} (\sigma(\mathbf{w}_{o_z} \pi(o_{z_i}) + b_{o_z}))$$

The hidden layers v_x and v_z have the same width h .

LTwin+RNN

This architecture, depicted in figure 4.2d, differs from the preceding one in that the max-pooling layer is replaced with a RNN.

Once again, RNN is an LSTM with linear activation: empirically, it has proved itself slightly more performant than a more conventional tanh activation followed by a fully connected layer.

Its output, parametrized by

$$\theta = \{\mathbf{w}_y, b_y, \mathbf{w}_{o_z}, b_{o_z}, \mathbf{w}_{o_x}, b_{o_x}, \mathbf{w}_z, b_z, \mathbf{w}_x, b_x\} \cup \{\mathbf{w}_{o_{RNN}}, \mathbf{w}_{RNN}\}$$

is:

$$f(x, \pi(o_x), \vec{z}, \pi(\vec{o}_z); \theta) = \mathbf{w}_y \begin{bmatrix} v_x \\ v_z \\ u_x \\ u_z \end{bmatrix} + \mathbf{b}_y$$

Where

$$v_x \equiv \sigma(\mathbf{w}_x \phi(x) + b_x)$$

and

$$v_z \equiv \text{loop}(\text{RNN}(\mathbf{w}_{\text{RNN}}), \langle l_1, \dots, l_m \rangle)$$

where

$$l_i \equiv \max_{i=1 \dots m} (\sigma(\mathbf{w}_z \phi(z_i) + b_z))$$

and

$$u_x \equiv \sigma(\mathbf{w}_{o_x} \pi(o_x) + b_{o_x})$$

and

$$u_z \equiv \text{loop}(\text{RNN}(\mathbf{w}_{o_{\text{RNN}}}), \langle j_1, \dots, j_m \rangle)$$

where

$$j_z \equiv (\sigma(\mathbf{w}_{o_z} \pi(o_{z_i}) + b_{o_z}))$$

The hidden layers v_x, v_z, u_x, u_z , have the same width h .

LTwin+2RNN

This architecture, depicted in figure 4.2d, differs from the preceding one in that the final fully connected layer is also replaced with a RNN.

Its output, parametrized by $\theta = \{\mathbf{w}_y, b_y, \mathbf{w}_{o_z}, b_{o_z}, \mathbf{w}_{o_x}, b_{o_x}, \mathbf{w}_z, b_z, \mathbf{w}_x, b_x\} \cup \{\mathbf{w}_{o_{\text{RNN}}}, \mathbf{w}_{\text{RNN}}, \mathbf{w}_{\text{RNN}}\}$, is and

$$f(x, \pi(o_x), \vec{z}, \pi(\vec{o}_z); \theta) = \text{loop}(\text{RNN}(\mathbf{w}_{\text{RNN}}), \langle v_x, v_z, u_x, u_z \rangle)$$

where v_x, v_z, u_x, u_z are defined as in LTwin+RNN.

LZIP

This architecture is depicted in figure 4.2c. Its output is:

$$f(x, \pi(o_x), \vec{z}, \pi(\vec{o}_z); \theta) = \text{loop}(\text{RNN}(\mathbf{w}_{\text{RNN}}), \langle v_x, u_x, v_{z_1}, u_{z_1}, \dots, v_{z_m}, u_{z_m} \rangle)$$

Where

$$v_x \equiv \sigma(\mathbf{w}_x \phi(x) + b_x)$$

and

$$u_x \equiv \sigma(\mathbf{w}_{o_x} \pi(o_x) + b_{o_x})$$

• — • ◊ • — •

Through the rest of this document we use abbreviations to denote the various models that arise through combinations of the various parameters detailed in this section, of the form:

$$\text{arch-m} \times \text{M-}\{+n:e_1\}\{+f:e_2\}$$

Where arch is the network architecture, m and M are the neighbourhood size and the max rank M , e_1 is the map applied to metadata for computing neighbourhoods used *if* it is different from id , e_2 is the mapping used to compute $\pi(\vec{o}_z)$ fed to the network *if* it is different from id .

Where m, M are obvious or are free parameters, we write

$$\text{arch-}\{+n:e_1\}\{+f:e_2\}$$

This page intentionally left blank.

Chapter 5

Experiments

5.1 Experimental protocol

We follow the protocol of [JBFF15] closely.

We use Tensorflow’s implementation of the RMSProp algorithm [Hin] with He-Zhang initialization [HZRS15], dropout with $p = 0.5$ batch size 64 (in lieu of 50, as found in [JBFF15]); we use $h = 500$, as in [JBFF15].

We use L_2 regularization with $\lambda = 3 \times 10^{-4}$ and learning rate 1×10^{-4} .

λ was chosen with grid search; learning failed to converge with $\lambda = 3 \cdot 10^{-3}$ reported in [JBFF15].

We use early stopping with a maximum of 10 and a minimum of 3 epochs, incremented to 15 and 5 for joint models; the results in [JBFF15] are obtained by selecting the best-performing model out of all 10 epochs instead.

These parameters, shared among all architectures, were chosen by coarse grid search and subsequent hand tuning.

The sheer number of architectures and parameters and the long running time of some experiments made it impractical to perform extensive grid search in the allotted time with the hardware available to us.

Attempting to hand-tune the hyperparameters for individual models was not found to be beneficial.

We run the models with (3, 6), (6, 12) and (12, 24) as choices of (m, M) .

5.1.1 Dataset and pretrained weights

We use the NUS-Wide dataset [CTH⁺09], which consists of a set of 269,648 images uploaded on the photo sharing website Flickr, annotated with ground truth for 81 concepts for evaluation.

NUS-WIDE is imbalanced over classes – whereas the tag “sky” is relevant for around 53,000 images, many classes have less than a thousand – or a hundred – images.

We restrict ourselves to the fixed subset of 190,253 images used in [JBFF15] for ease of comparison; the images used are those that were still online when the authors of [JBFF15] conducted their experiments. They are tagged with a matrix of 422,364 unique Flickr tags, which we narrow down to the $\tau = 5000$ most frequent tags.

The dataset is split into a training set, a validation set and a test set.

We build “splits”, in which images are randomly partitioned to form test, validation and test sets of 110,000, 40,000 and 40,253 images respectively.

We generate 5 such splits; we run all experiments on all 5 splits and average the results.

We only train the last layer of the CNN; our CNN of choice is the the implementation of AlexNet[KSH12] found in Caffe, that comes with pretrained weights for ImageNet [DDS⁺09], same as [JBFF15].

5.1.2 Semantic mappings and distance measures

Given a map or dictionary of *embeddings* $\beta : \text{TAGS} \rightarrow \mathbb{R}^n$ for some n , we define $\rho(o_x; \beta)$ as the sum of the vectors $\beta(t_{(i)})$ for each tag $t_{(i)}$ relevant for x , i.e.:

$$\rho(o_x; \beta) \triangleq \sum_{i=1}^{\tau} o_{x(i)} \cdot \beta(t_{(i)}) \quad (5.1)$$

We consider the following possibilities for mappings π and distance measures δ , which we will discuss in detail shortly:

1. $\pi \equiv \text{id}$, $\delta \equiv \mathcal{J}$, where \mathcal{J} is the Jaccard distance
2. $\pi(x) \equiv \rho(o_x; \beta)$ and $\delta \equiv \text{sim}_{\cos}$ where β can be:
 - (a) w2v
 - (b) wnet
 - (c) synthwnet

id – i.e. raw binary vectors

Neighbourhoods are computed using the Jaccard distance \mathcal{J} (equation (3.1)) between binary vectors as a distance measure.

Binary vectors o_x for each image x (or neighbour z_i) are directly handled by the neural network, without further processing.

w2v

Recall from section 2.4 that w2v vectors are designed to be similar for words that have a high co-occurrence frequency and, therefore, are likely to be semantically similar.

We use a dictionary of word2vec embeddings made available from [w2v19]; they were obtained by training on a 100-billion-words subset of the Google News database; it contains 300-dimensional vectors for 3 million words and phrases.

In this way we expect (see discussion of w2v in section 2.4) to recover some semantic information from the tags and improve performance – as well as achieving decoupling from the low-level binary representation for joint architectures (section 4.2).

Whereas previously an image tagged only with boy and motorcycle would have empty intersection and therefore infinite distance from one tagged with kid and motorbike, we expect the vectors $w2v(\text{boy})$ (resp. $w2v(\text{motorcycle})$) to be spatially “similar” to $w2v(\text{kid})$ (resp. $w2v(\text{motorbike})$).

We expect the sum of similar vectors to yield a similar vector, and, therefore, images tagged with semantically similar sets of tags to produce similar sums ρ .

Cosine distance is the choice of δ used, as defined as [MRS08]:

$$\text{dist}_{\cos}(x_1, x_2) = 1 - \frac{\vec{x}_1 \cdot \vec{x}_2}{|\vec{x}_1||\vec{x}_2|}$$

Notice that cosine similarity is insensitive to the absolute magnitude of vectors – we don't expect a significant difference between computing cosine similarity over the sum equation (5.1) of individual embeddings vs. their average.

wnet

WordNet embeddings work in the same fashion as w2v embeddings, except that $\beta(x)$ is extracted from a dictionary where vector representations are optimized to be similar if the words are close on the WordNet ontological graph.

Cosine distance is again the choice of δ .

WordNet embeddings are those made available by the authors of [SBRS] on their repository [NG19], a dictionary of 650-dimensional vectors obtained from Princeton WordNet 3.0 with 60,000 words.

Recalling that our vocabulary has size $\tau = 5000$, we note that 726,722 total tags and 1432 unique tags (0.2864 of τ) are not present in the w2v dictionary, whereas 1,791,289 total and 3513 unique tags (0.7026 of τ) are not present in the wnet dictionary.

We therefore expect the WordNet dictionary to be at a very significant disadvantage.

synthwnet embeddings

We observe that the cardinality of intersection set between the w2v and the wnet dictionary is 26,887 words, which is not large, but an order of magnitude larger than the intersection between the wnet dictionary and the tag vocabulary.

We therefore attempt a knowledge transfer approach and impute some of the missing vectors by training a neural network to approximate the missing wnet vector given the w2v vector that we have.

We don't expect, of course, to make the missing information appear out of nowhere – we just want to leverage the wnet vectors that we do have; the imputed ones will embed *at most* as much information as the w2v vectors that were used to predict them.

We use a feedforward network with two hidden layers of respectively 2000 and 1500 units, batch size 64, learning rate 0.02 and we train it with backpropagation for 50 epochs, evaluating it every 10 epochs over a validation set made of 25% of the elements.

The model achieved a MSE of 0.00416 and was used to estimate vectors for 2101 tags that exclusively appear in the w2v dictionary.

We call the resulting dictionary synthwnet.

5.2 Metrics

We report per-class and per-image mAP, prec@3, rec@3 as defined in section 2.3.1.

Notice that prec@k, rec@k make the assumption a *fixed* number k of predicted labels is relevant for every image.

	prec@3	rec@3
Per-label	60.68±1.32	68.52±0.35
Per-image	92.09±0.10	66.83±0.12

Table 5.1: Upper bounds for our dataset [JBFF15]

Therefore, if the number of relevant labels is not constant k in the ground truth, it is impossible to achieve precision (resp. recall) of 1, if there are images with less (resp. more) than k relevant labels.

This is the case with our dataset: the upper bounds for $k = 3$ on our dataset are shown in table 5.1.

The theoretical upper bound for mAP_{img} and mAP_{label} is 1.0, but we have strong reason to doubt that it can be achieved in practice, due to a certain randomness in the tags introduced by the individual leanings of the humans who provided ground truth for our dataset (see again the comments to figure 4.3).

5.3 Results

Full results are summarized in table 5.2, table 5.3, table 5.4.

We choose to focus our attention, for the most part, on mean mAP_{label} and mAP_{img} , since they summarize separate precision and recall metrics.

Firstly, we note that mAP_{label} is the metric that is affected the most *in general*, whereas mAP_{img} remains more stationary.

This is made evident in figure 5.1.

5.3.1 Neural architectures

Baseline: Visual only

We report the performance of the visual-only classifier separately, in table 5.5.

Visual architectures: LTN vs RTN

The performance for the LTN and RTN architectures, with or without π other than id used during neighbourhood generation, is plotted against neighbourhood size in figure 5.3.

For the same neighbourhoods, RTN leads to an improvement mAP_{label} of around 0.7 to 1.2 percentage points over LTN, in exchange for a drop of 0.2 to 0.4 percentage points in mAP_{img} .

More interestingly, the gap between $\pi = id$ and $w2v$ is larger for RTN at low values of m .

Notice how RTN with $w2v$ embeddings and a 3×6 neighbourhood outperforms “vanilla” LTN with a 6×12 neighbourhood in terms of mAP_{label} , with negligible impact on mAP_{img} .

The performance of RTN begins to decline faster than LTN with $\pi = wnet$, and is particularly bad with *synthwnet*.

This leads to hypothesize that RTN is particularly sensitive to the quality of neighbourhoods it is trained on.

All models improve monotonically with m .

arch	neighbours	feed	mAP _{label}	mAP _{img}	rec _{label}	prec _{label}	rec _{img}	prec _{img}
ltn	id		48.05±0.16	78.34±0.19	42.11±0.82	44.24±1.28	72.74±0.22	52.33±0.19
ltn	w2v		48.52±0.21	78.50±0.16	42.29±1.12	44.79±0.98	72.89±0.15	52.44±0.13
ltn	wnet		49.32±0.19	★ 78.66±0.12	42.76±1.13	45.42±1.02	73.10±0.12	52.60±0.12
ltn	synth		48.31±0.23	78.30±0.16	41.96±0.89	44.99±1.05	72.72±0.13	52.33±0.16
rtn	id		48.78±0.26	77.92±0.26	44.38±1.41	46.39±1.64	72.45±0.21	52.04±0.25
rtn	w2v		50.26±0.27	78.23±0.14	45.93±1.56	46.93±2.06	72.69±0.13	52.20±0.15
rtn	wnet		★ 50.58±0.30	78.40±0.24	46.15±2.05	47.30±2.69	72.92±0.18	52.35±0.25
rtn	synth		46.97±0.24	77.85±0.21	41.76±1.29	44.91±1.11	72.27±0.18	52.02±0.19
ltn-vec	id		51.60±0.08	80.47±0.13	43.88±0.93	47.33±1.10	74.76±0.13	53.79±0.14
ltn-allvecs	id		53.52±0.06	80.55±0.22	46.01±1.50	47.46±1.57	74.86±0.16	53.81±0.18
lzip	id	id	61.03±0.23	82.85±0.18	53.78±1.56	51.09±1.95	77.13±0.06	55.28±0.15
lzip	w2v	id	61.55±0.34	82.85±0.26	54.86±2.35	50.85±3.29	77.18±0.20	55.25±0.23
lzip	w2v	w2v	62.66±0.29	82.90±0.28	56.53±2.15	51.05±2.86	77.26±0.24	55.25±0.25
lzip	id	w2v	62.66±0.29	82.90±0.28	56.53±2.15	51.05±2.86	77.26±0.24	55.25±0.25
ltwin	id	id	56.07±0.17	82.51±0.10	47.88±1.20	49.07±1.47	76.80±0.12	55.15±0.10
ltwin	id	w2v	63.55±0.27	★ 83.83±0.08	53.99±1.15	52.74±1.37	78.17±0.05	55.87±0.13
ltwin	w2v	id	56.74±0.24	82.87±0.11	48.18±1.19	50.08±1.27	77.19±0.12	55.40±0.16
ltwin	w2v	w2v	★ 63.69±0.20	83.82±0.10	54.36±1.25	52.67±1.10	78.17±0.05	55.88±0.17
ltwin	id	wnet	52.70±0.23	80.88±0.13	45.80±1.13	46.86±1.58	75.25±0.10	54.08±0.14
ltwin	wnet	id	56.48±0.30	82.65±0.10	47.82±1.25	49.86±1.20	76.98±0.09	55.28±0.11
ltwin	wnet	wnet	52.94±0.40	80.86±0.11	45.71±1.30	46.90±1.73	75.23±0.10	54.06±0.13
ltwin	id	synth	57.23±0.21	82.24±0.10	48.64±0.96	49.78±0.61	76.64±0.06	54.94±0.12
ltwin	synth	id	56.34±0.12	82.73±0.08	47.65±1.18	50.19±1.31	77.00±0.09	55.30±0.12
ltwin	synth	synth	57.17±0.30	82.19±0.11	48.66±1.39	49.91±1.30	76.58±0.11	54.90±0.15
ltwin-rnn	id	id	55.05±0.11	82.29±0.10	47.22±1.60	48.60±2.24	76.56±0.11	55.00±0.14
ltwin-2rnn	id	id	60.38±0.32	80.25±2.56	50.47±3.63	50.05±5.30	75.04±2.29	53.54±1.80
ltwin-2rnn	id	w2v	61.75±0.73	82.34±0.31	51.58±1.67	48.97±2.34	76.99±0.25	54.98±0.28
ltwin-2rnn	w2v	w2v	60.92±0.48	82.37±0.30	53.74±1.36	47.01±1.69	76.86±0.15	54.91±0.19
ltwin-2rnn	id	wnet	60.38±0.75	81.99±0.73	50.98±1.63	48.44±2.40	76.55±0.53	54.66±0.58

Table 5.2: Summary of results for neighbourhood size $(m, M) = (3, 6)$

arch	neighbours	feed	mAP _{label}	mAP _{img}	rec _{label}	prec _{label}	rec _{img}	prec _{img}
ltn	id		50.20±0.21	79.02±0.12	43.71±1.02	46.23±1.35	73.45±0.08	52.82±0.13
ltn	w2v		51.44±0.19	* 79.45±0.11	44.47±1.38	46.63±1.74	73.91±0.09	53.10±0.15
ltn	wnet		51.14±0.23	79.35±0.14	44.53±1.29	46.57±1.12	73.80±0.10	53.04±0.15
ltn	synth		50.15±0.24	78.88±0.17	43.58±1.36	45.85±1.59	73.35±0.15	52.70±0.15
rtn	id		51.79±0.19	78.70±0.32	46.13±2.05	48.84±3.02	73.18±0.21	52.57±0.29
rtn	w2v		* 53.55±0.37	79.13±0.13	48.13±2.41	49.50±2.78	73.60±0.12	52.79±0.18
rtn	wnet		52.61±0.22	79.00±0.37	47.21±2.19	49.18±2.84	73.52±0.22	52.79±0.33
rtn	synth		48.29±0.24	78.07±0.41	42.72±1.58	45.65±1.63	72.54±0.33	52.16±0.30
ltn-vec	id		52.63±0.26	80.75±0.18	44.36±1.08	47.98±1.48	75.08±0.16	54.00±0.14
ltn-allvecs	id		54.82±0.08	80.93±0.21	46.90±1.34	47.89±1.72	75.28±0.13	54.07±0.17
lzip	id	id	61.40±0.25	82.56±0.17	54.84±2.31	51.03±2.82	76.86±0.14	55.05±0.17
lzip	w2v	id	62.12±0.53	82.90±0.31	55.23±1.51	51.16±1.99	77.28±0.26	55.31±0.28
lzip	w2v	w2v	62.33±0.16	82.91±0.18	54.78±2.12	50.65±1.90	77.34±0.13	55.29±0.21
lzip	id	w2v	62.33±0.16	82.91±0.18	54.78±2.12	50.65±1.90	77.34±0.13	55.29±0.21
ltwin	id	id	56.16±0.26	82.53±0.14	47.71±1.41	49.31±1.20	76.86±0.13	55.16±0.14
ltwin	id	w2v	* 63.38±0.31	* 83.87±0.09	54.15±1.30	52.10±1.51	78.17±0.08	55.86±0.09
ltwin	w2v	id	57.09±0.14	82.97±0.11	48.36±1.24	50.02±1.25	77.25±0.13	55.43±0.13
ltwin	w2v	w2v	63.34±0.42	83.77±0.14	54.57±1.44	52.33±1.42	78.13±0.09	55.84±0.19
ltwin	id	wnet	53.22±0.19	81.11±0.08	46.09±1.55	46.69±1.68	75.45±0.10	54.20±0.12
ltwin	wnet	id	56.54±0.09	82.69±0.10	47.88±0.93	50.32±1.26	76.93±0.09	55.25±0.14
ltwin	wnet	wnet	53.41±0.20	80.97±0.15	46.21±1.06	48.51±1.03	75.29±0.13	54.09±0.16
ltwin	id	synth	57.23±0.10	82.33±0.14	48.74±1.28	49.65±1.06	76.71±0.14	54.99±0.13
ltwin	synth	id	56.77±0.24	82.78±0.18	47.93±1.69	50.41±1.77	77.09±0.21	55.33±0.17
ltwin	synth	synth	56.97±0.35	82.19±0.08	48.90±1.49	49.83±1.31	76.61±0.13	54.90±0.15
ltwin-rnn	id	id	56.28±0.19	82.57±0.16	48.25±1.61	49.39±2.23	76.85±0.17	55.16±0.14
ltwin-2rnn	id	id	61.65±0.55	81.69±0.60	52.95±2.38	48.75±2.16	76.30±0.42	54.48±0.51
ltwin-2rnn	id	w2v	62.50±0.54	82.86±0.38	53.17±2.14	50.67±2.85	77.37±0.21	55.24±0.25
ltwin-2rnn	w2v	w2v	61.87±0.33	82.35±0.30	54.43±2.00	49.44±2.40	76.90±0.09	54.91±0.15
ltwin-2rnn	id	wnet	61.45±0.22	82.32±0.30	51.54±2.56	49.43±3.35	76.81±0.30	54.89±0.23

Table 5.3: Summary of results for neighbourhood size $(m, M) = (6, 12)$

arch	neighbours	feed	mAP _{label}	mAP _{img}	rec _{label}	prec _{label}	rec _{img}	prec _{img}
ltn	id		53.17±0.12	79.82±0.16	45.67±1.75	47.64±2.18	74.29±0.13	53.34±0.17
ltn	w2v		54.54±0.13	* 80.32±0.16	47.41±1.56	48.57±2.34	74.83±0.14	53.64±0.21
ltn	wnet		53.07±0.17	79.95±0.24	46.09±1.86	47.41±1.66	74.35±0.16	53.38±0.19
ltn	synth		52.81±0.10	79.46±0.14	45.57±1.93	47.28±2.03	73.95±0.09	53.08±0.13
rtn	id		53.97±0.27	79.23±0.27	48.44±3.24	48.75±4.01	73.67±0.25	52.85±0.36
rtn	w2v		* 55.36±0.34	79.77±0.27	48.73±2.77	51.21±2.61	74.35±0.29	53.28±0.24
rtn	wnet		53.76±0.33	79.45±0.30	47.51±2.38	50.40±2.59	73.87±0.24	53.03±0.26
rtn	synth		50.22±0.46	78.49±0.25	43.41±1.77	46.96±2.21	72.92±0.28	52.43±0.22
ltn-vec	id		54.86±0.20	81.34±0.15	46.56±1.39	50.10±1.70	75.67±0.17	54.37±0.14
ltn-allvecs	id		56.61±0.12	81.28±0.21	48.18±1.48	48.93±2.32	75.66±0.14	54.29±0.19
lzip	id	id	60.64±0.14	82.42±0.32	52.53±1.67	51.43±2.36	76.65±0.23	54.91±0.26
lzip	w2v	id	61.24±0.51	82.36±0.41	53.70±2.58	49.86±2.46	76.79±0.26	54.99±0.16
lzip	w2v	w2v	60.19±0.57	82.32±0.15	52.76±2.09	49.47±2.53	76.78±0.25	54.91±0.10
lzip	id	w2v	62.33±0.16	82.91±0.18	54.78±2.12	50.65±1.90	77.34±0.13	55.29±0.21
ltwin	id	id	56.79±0.24	82.64±0.08	48.73±1.47	49.22±1.70	76.93±0.13	55.21±0.14
ltwin	id	w2v	* 63.13±0.31	* 83.77±0.06	54.40±1.33	51.86±1.58	78.06±0.05	55.78±0.13
ltwin	w2v	id	57.73±0.17	83.00±0.06	49.24±1.67	50.73±2.19	77.30±0.13	55.43±0.09
ltwin	w2v	w2v	63.09±0.16	83.70±0.14	54.80±1.67	51.86±1.63	78.04±0.08	55.76±0.16
ltwin	id	wnet	55.12±0.25	81.48±0.10	47.50±1.64	48.37±1.69	75.86±0.09	54.43±0.11
ltwin	wnet	id	56.83±0.24	82.64±0.10	48.48±1.42	50.38±1.66	76.88±0.11	55.18±0.13
ltwin	wnet	wnet	54.01±0.14	81.06±0.10	46.87±1.41	47.78±1.36	75.40±0.09	54.12±0.18
ltwin	id	synth	57.58±0.15	82.36±0.09	48.93±1.43	50.65±2.11	76.72±0.12	54.98±0.09
ltwin	synth	id	57.69±0.39	82.82±0.12	48.80±1.37	50.17±1.30	77.13±0.13	55.34±0.08
ltwin	synth	synth	57.32±0.16	82.22±0.08	49.35±1.67	50.01±1.62	76.65±0.11	54.93±0.14
ltwin-rnn	id	id	58.87±0.43	82.95±0.08	50.50±1.42	51.55±1.31	77.19±0.12	55.36±0.12
ltwin-2rnn	id	id	62.00±1.44	80.52±2.79	51.04±5.28	51.15±3.52	75.37±2.40	53.73±1.97
ltwin-2rnn	id	w2v	63.04±0.22	83.02±0.34	53.08±1.40	50.33±1.97	77.45±0.25	55.30±0.39
ltwin-2rnn	w2v	w2v	62.33±0.33	82.72±0.37	54.58±2.70	51.17±3.51	77.07±0.27	55.07±0.26
ltwin-2rnn	id	wnet	62.35±0.56	82.56±0.26	51.43±2.44	52.11±1.76	77.17±0.13	55.15±0.18

Table 5.4: Summary of results for neighbourhood size $(m, M) = (12, 24)$

arch	mAP _{label}	mAP _{img}	rec _{label}	prec _{label}	rec _{img}	prec _{img}
v-only	45.05 ± 0.11	76.88 ± 0.11	42.31 ± 0.59	43.74 ± 1.07	71.41 ± 0.13	51.36 ± 0.13

Table 5.5: Results for v-only

It is worth noting that we are using the RNN for its properties as an universal computer (see discussion in section 2.2.2) rather than purely to learn a function over sequential data.

In fact, for a sufficiently large dataset, given that the correlation between metadata and ground truth is rather loose, we’d expect not to see a significance in the ordering of the nearest m neighbours for practical values of m .

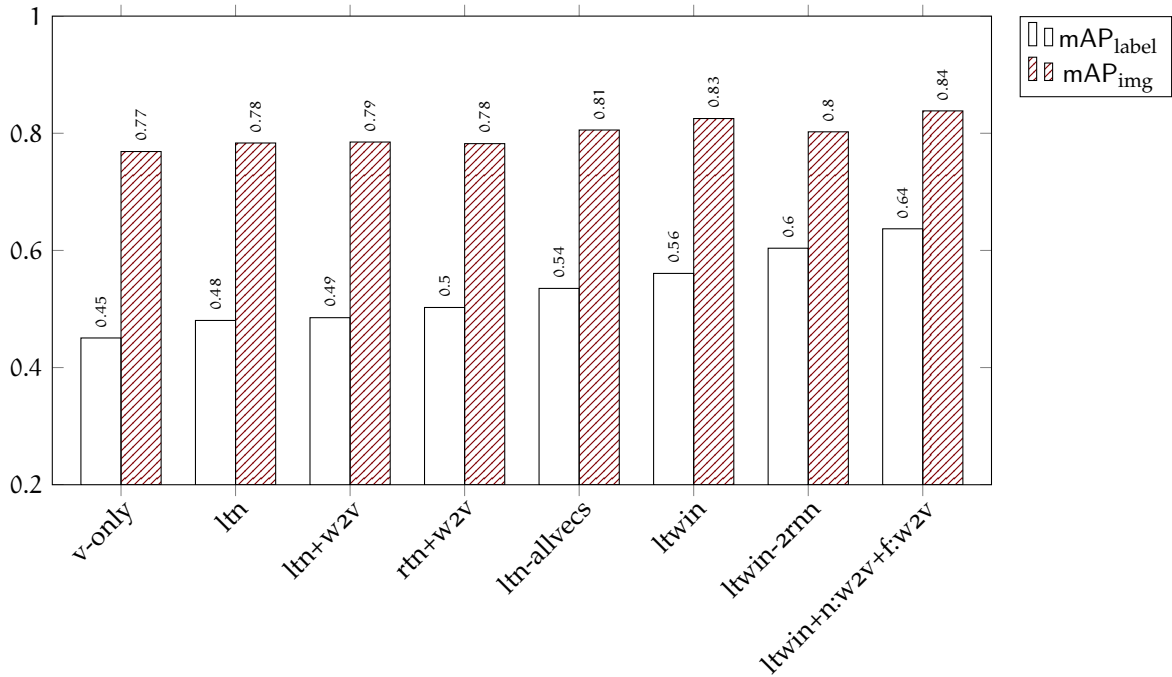


Figure 5.1: Mean mAP_{label} and mAP_{img} of select models showing the spectrum with $(m, M) = (3, 6)$

Joint neural architectures: LTN+AllVecs, LTwin,LZIP, LTwin+RNN, LTwin+2RNN

Tag vectors are, on their own, a powerful predictor, as seen in [ML12], and the immediate availability of raw metadata – rather than implicit knowledge embedded in visual features from neighbours – can lead to a significant boost in performance.

We do not wish, therefore, to directly compare joint architectures that are directly fed metadata against purely visual architectures – however, there is a greater disparity in performance between these architectures that we wish to explore than there is between LTN and RTN.

This is a function of *how* the available metadata information is used through the architecture and the semantic mappings that are applied to the data; this is visualized in figure 5.2 and in figure 5.5.

Without semantic mapping Consider first what happens in the “naive” case, i.e. with $\pi = \text{id}$.

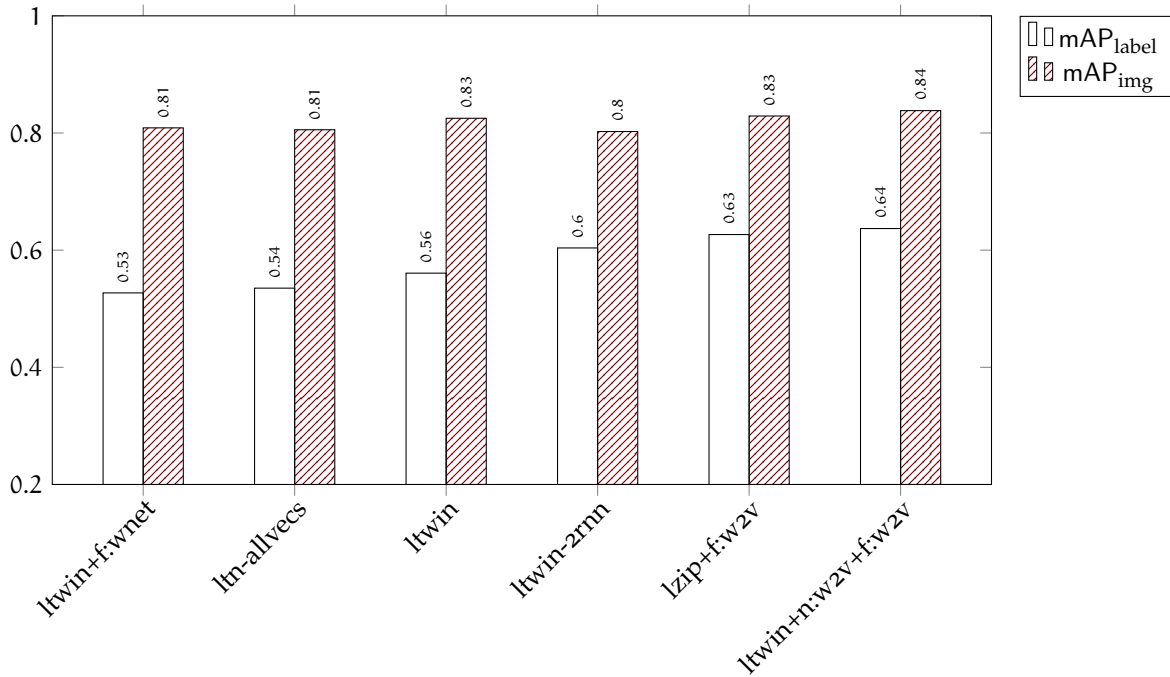


Figure 5.2: Mean mAP_{label} and mAP_{img} of select joint models showing the spectrum with $(m, M) = (3, 6)$

As seen in figure 5.4, the simplest and worst-performing model uses LTN+AllVecs as an architecture and is fed raw binary vectors; it shows quasi-linear improvement wrt neighbourhood size.

LZIP – which uses a RNN – improves uniformly upon it and achieves very good mAP_{label} and mAP_{img} from the start but tends to exhibit a mild decrease in performance with neighbourhood size, along with LTwin+2RNN.

In turn, LTwin achieves good mAP_{img} but comparatively poor mAP_{label} ; LTwin+RNN achieves roughly comparable performance, but shows linear improvement with m .

LTN+AllVecs is the clear loser, whereas LZIP – at small (m, M) – and LTwin+2RNN are the best-performing models, with LTwin comfortably in the middle.

Unfortunately, LZIP and LTwin+2RNN are also by far the longest to train by an order of magnitude (we just need to consider the breadth of the unrolled graph for non-trivial neighbourhood sizes to persuade ourselves this is the case).

LZIP, LTwin, LTwin+RNN with metadata semantic mappings As seen in figure 5.5 and figure 5.6, the addition of semantic metadata transforms can give a significant boost to performance, *in addition* to the benefits wrt robustness of the model to vocabulary changes and applicability to a different database than the one used for training mentioned in section 4.2.

The performance of all architectures is boosted when they are fed transformations computed from w2v vectors through equation (5.1) instead of plain binary vectors – recall that our dictionary covers less than 75% of the tag vocabulary, so there might be further room for improvement.

All models tend to saturate around $(\text{mAP}_{\text{label}}, \text{mAP}_{\text{img}}) = (.63, .83)$; it is unclear if we're hitting an intrinsic performance limit on our dataset (recall again the discussion in section 5.2).

This appears to be the case particularly for LZIP, even without any sort of π .

It may be the case that the simpler LTwin can match the performance of the more complex models once provided with w2v mappings.

ltwin+f:w2v performs as well as ltwin+n:w2v+f:w2v, or even better; the same goes for its LZIP siblings (by a considerably minor margin).

We speculate that the ability of the network to learn to take maximal advantage of semantic embeddings overshadows the effect of their use in neighbourhood generation and using w2v vectors in the neighbourhood generation process might therefore be unnecessary.

lzip+f:w2v, ltwin+f:w2v emerge as the superior models.

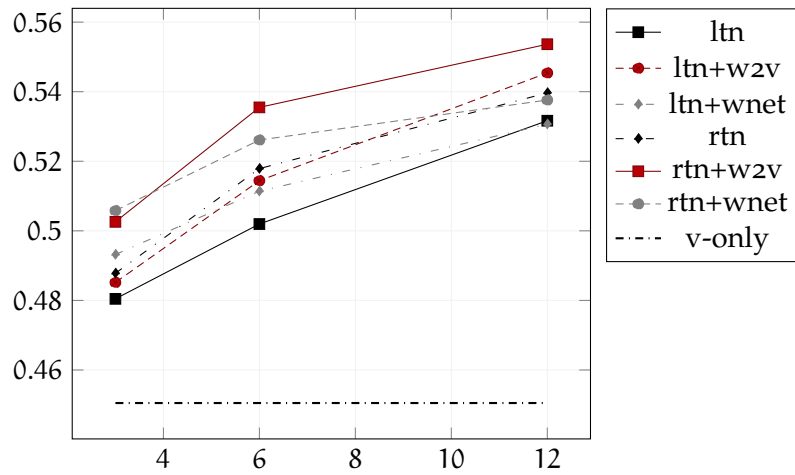
With wnet and synthwnet The results of wnet and synthwnet are compared in figure 5.7. As expected, wnet results in poor performance.

Notice also how ltwin+f:wnet is particularly sensitive to neighbourhood size.

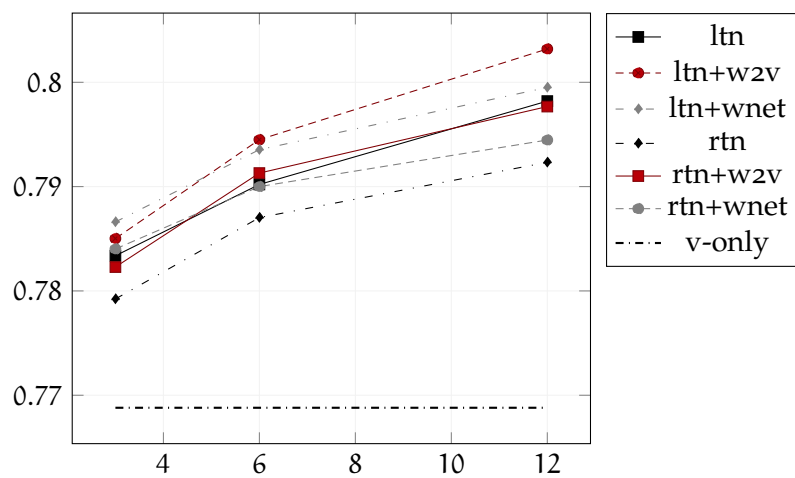
It appears that using the synthwnet dictionary leads to a uniform improvement for LTwin, which is particularly evident when the transformed metadata is fed to the network instead of raw binary vectors.

The improvement is marginal or nil for LTN, RTN, suggesting that the additional information is not sufficient to yield better-quality neighbourhoods.

Nevertheless, this suggests that this form of transfer learning can be effective, particularly when using joint models where we have two dictionaries with a large intersection between each other but a small and non-overlapping intersection between each and the tag vocabulary.

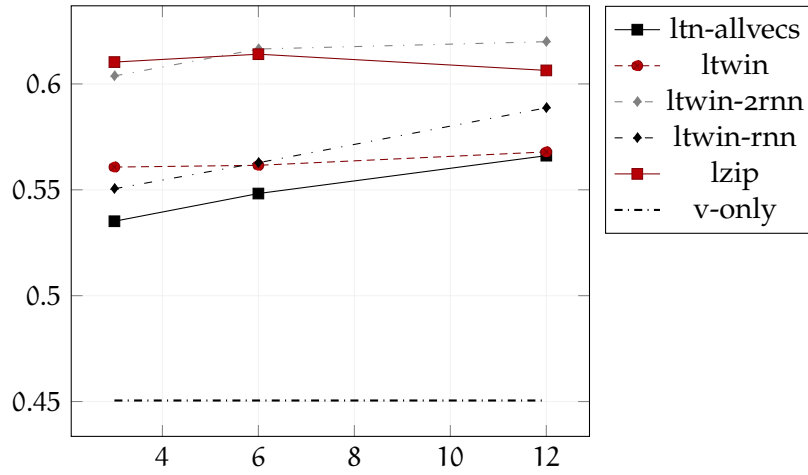


(a) Mean mAP_{label} vs m

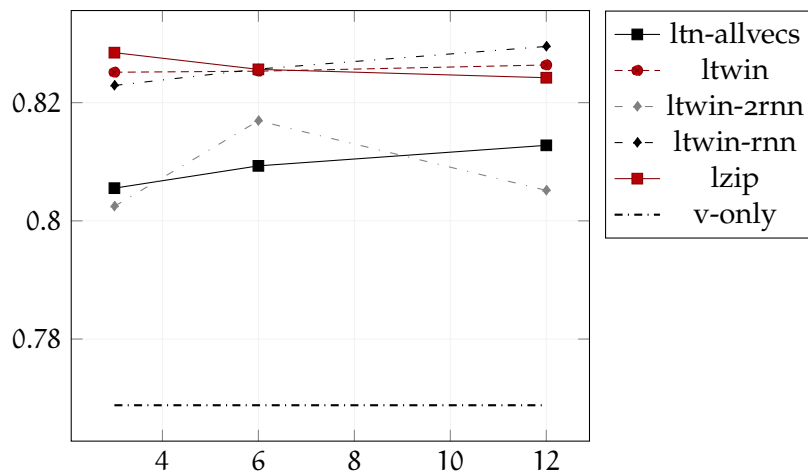


(b) Mean mAP_{img} vs m

Figure 5.3: Mean mAP_{img} and mAP_{label} vs m for visual neural architectures

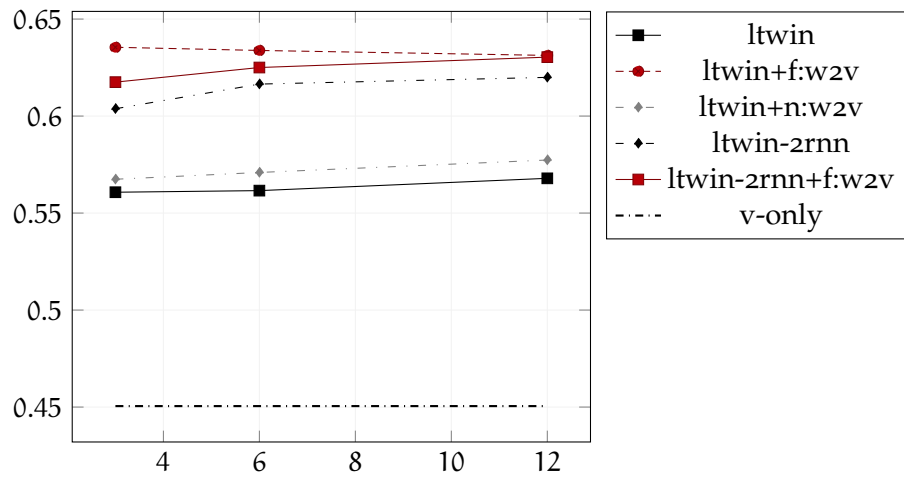


(a) Mean mAP_{label} vs m

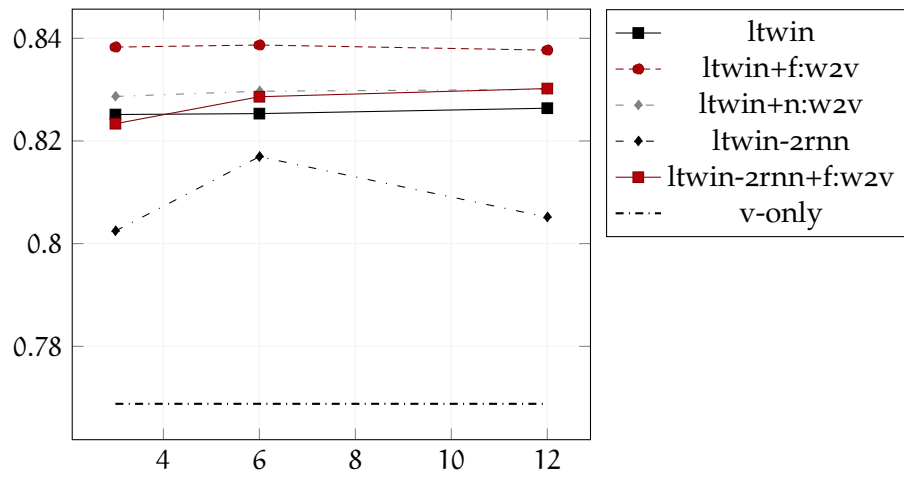


(b) Mean mAP_{img} vs m

Figure 5.4: Mean mAP_{img} and mAP_{label} vs m for select joint models under with $\pi = \text{id}$

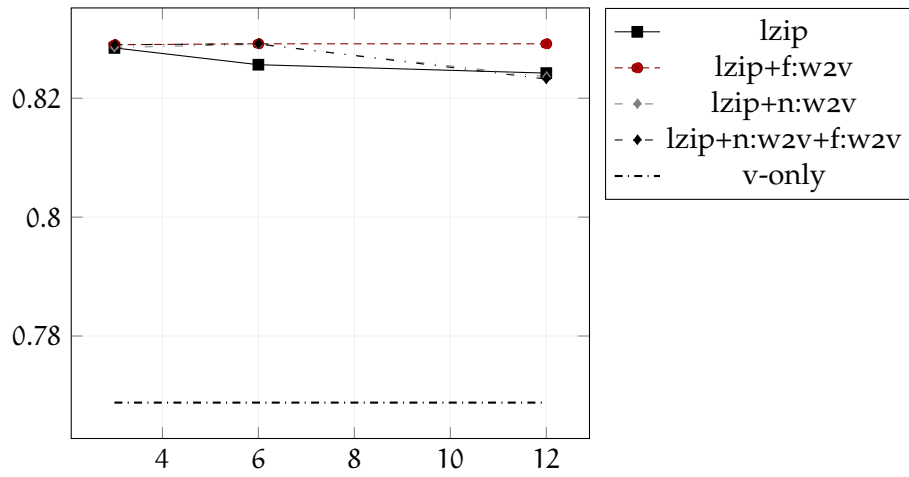


(a) Mean mAP_{label} vs m

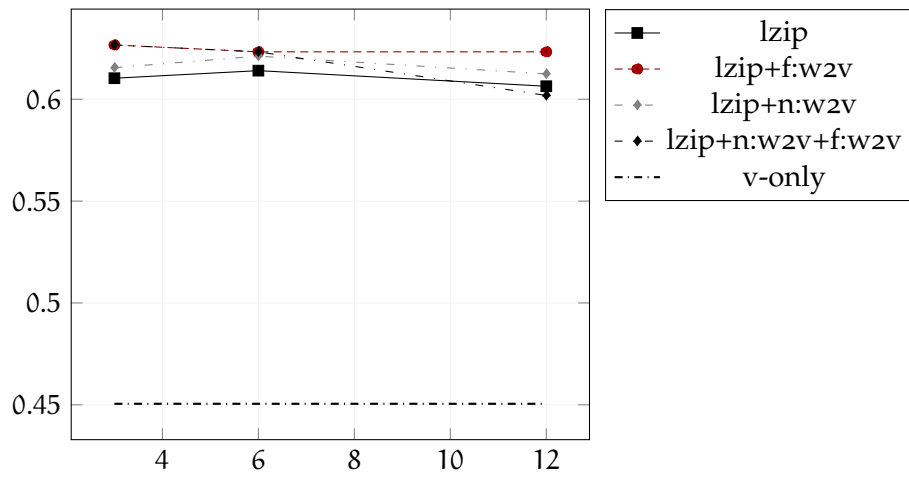


(b) Mean mAP_{img} vs m

Figure 5.5: Mean mAP_{img} and mAP_{label} vs m for select joint models, various choices of π



(a) Mean mAP_{img} vs m



(b) Mean mAP_{label} vs m

Figure 5.6: Mean mAP_{img} and mAP_{label} vs m for LZIP. Note the relative indifference to π as opposed to figure 5.5

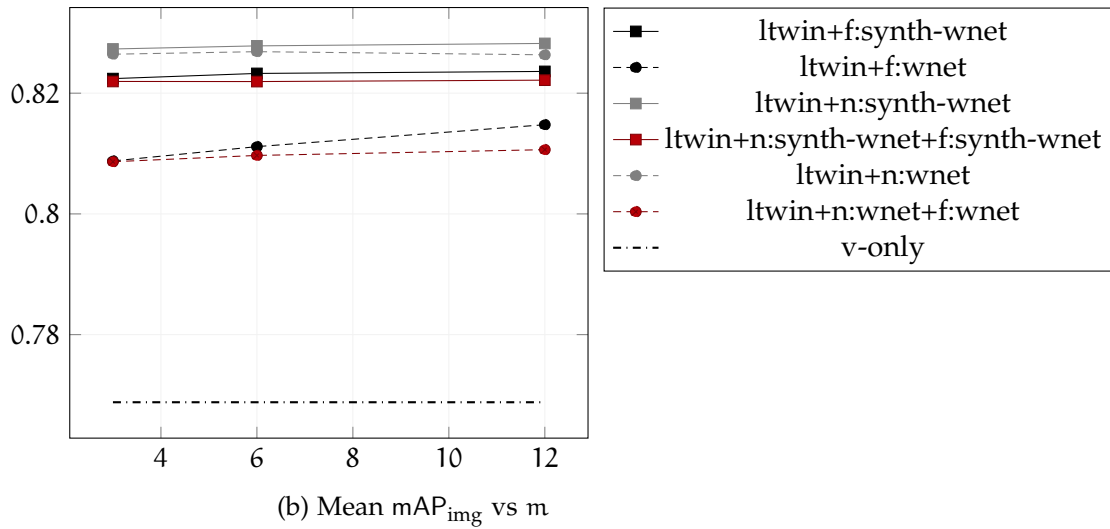
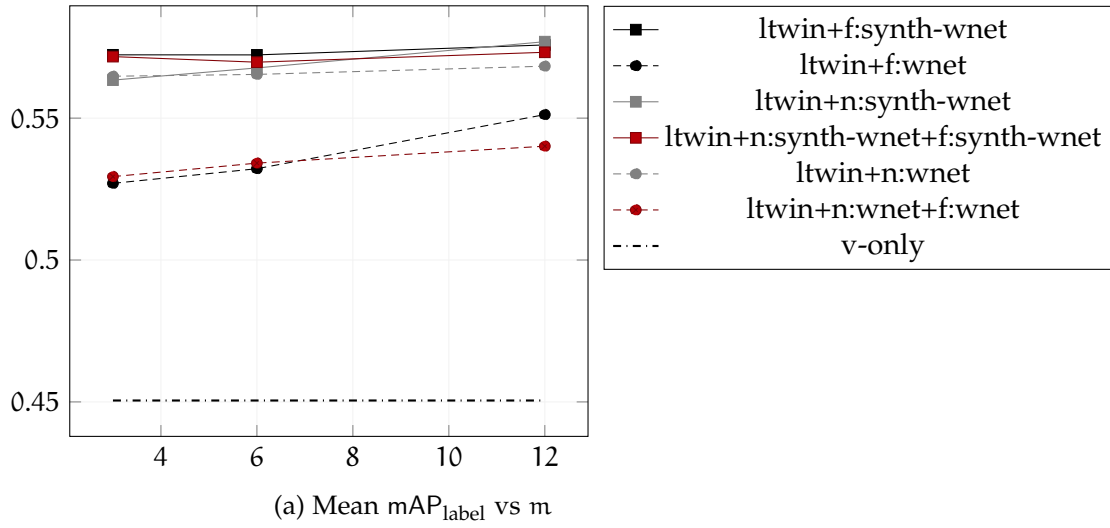
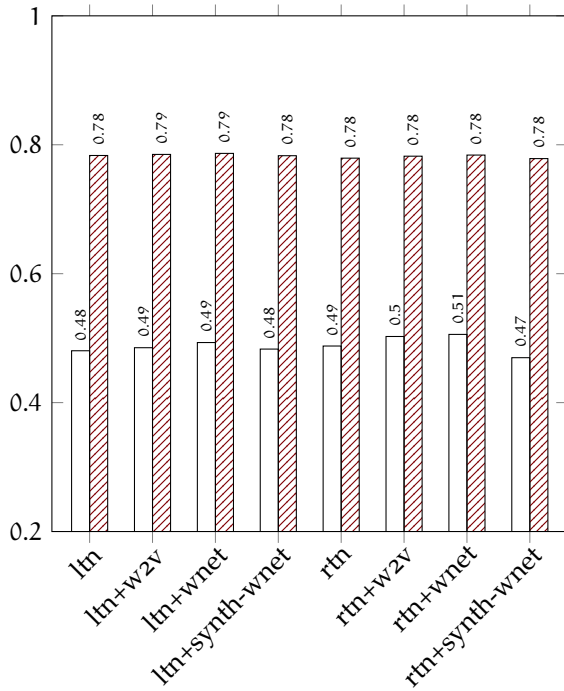
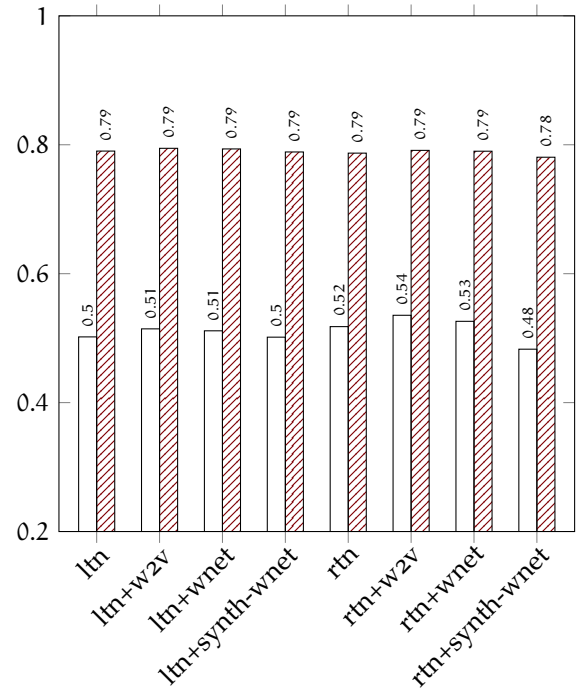


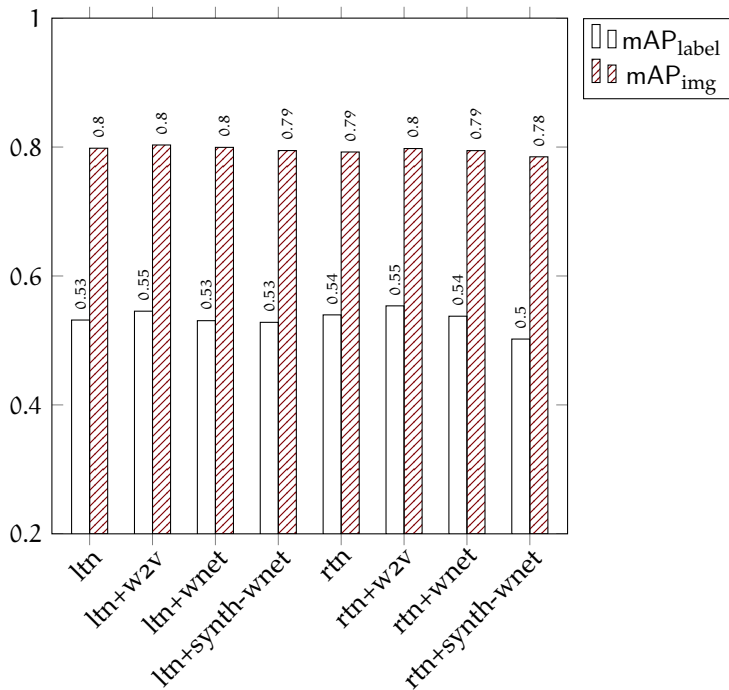
Figure 5.7: Mean mAP_{label} and mAP_{img} of LTwin with $\pi = \text{wnet}$ and $\pi = \text{synthwnet}$



(a) with neighbourhood size $(m, M) = (3, 6)$



(b) with neighbourhood size $(m, M) = (6, 12)$



(c) with neighbourhood size $(m, M) = (12, 24)$

Figure 5.8: Mean mAP_{label} and mAP_{img} of LTN, RTN at various neighbourhood sizes and metadata mappings

Chapter 6

Variations upon the experimental protocol and future work

Results given in section 5.3 were obtained by matching the protocol and implementation of [JBFF15], for ease of comparison with the baseline.

This in particular implies using the *entire* dataset as a source of neighbours both at train and test time.

However, this does not measure how truly robust a model is to changes in the underlying database, i.e. it doesn't measure its potential to be deployed on a different database than the one it was trained on.

This is because there is a possibility, *which neighbourhood randomization is intended to avert*, that the models overadapts to contingent features of the neighbours and fails to generalize to the visual features of entirely new neighbours.

Existing literature (see e.g. the review in [ZWZ⁺19]) uses frequently a protocol in which the dataset is split in two or three subsets *closed under neighbours*, with proportions approximating the classic 60:20:20 ratio, depicted in figure 6.7.

However, we do not believe that this is an entirely “unbiased” protocol: at training time the neighbours are drawn from a significantly larger subset. We expect the average distance to be greater in the smaller subset¹

Intuitively, this can mean noisier neighbourhoods at test time, particularly if the test and validation sets are small.

The intuition is given in figure 6.1.

We ran some experiments on a subsection of models with this setup, extracting neighbours at training, validation and test time from the same 110000, 40000 and 40253 images that make up the respective set.

This means that now the neighbours are extracted from pools of cardinality respectively less than $\frac{1}{2}$, $\frac{1}{6}$ and $\frac{1}{6}$ times the size of the whole dataset.

The results are given in table 6.3, table 6.1, table 6.2 and visualized in figure 6.5, figure 6.3 figure 6.4.

We see a partly different picture: while the proportions between our models are roughly the same at $(m, M) = (3, 6)$, it is the case that as neighbourhood size increases, performance

¹We believe a proof can be worked out with a graph-theoretic approach, by thinking of removing an image from the pool as removing a vertex and replacing it with edges between its neighbours, then using the results of [BDD⁺12] [Cab17]

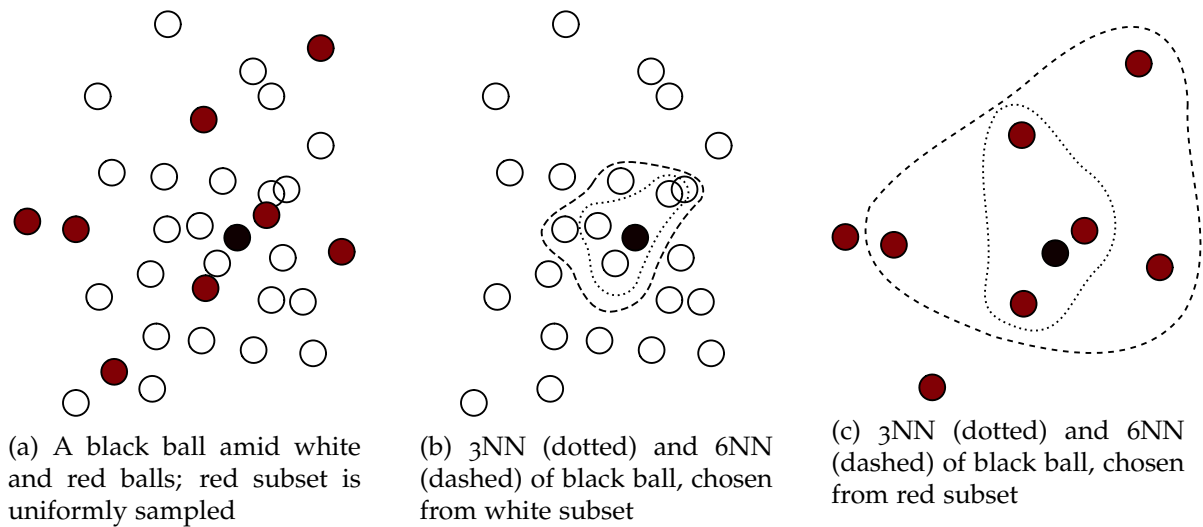


Figure 6.1: The expected distance of nearest neighbours is larger if chosen from a strictly smaller, uniformly sampled subset of the same set

gets worse, even significantly in the case of LTN.

At $(m, M) = (3, 6)$ LTN performs only mildly better than the visual-only classifier.

RTN is worse on average but shows less variation – recall that unlike LTN it is able to distinguish the different neighbours and may easily learn to “distrust” the farther ones, making the results consistent with the hypothesis of noisy neighbourhoods.

This is also the case for LZIP, leading to the conjecture that RNN-based architectures need a larger and diverse neighbour pool to fully realize their potential.

We believe, however, that the truly “unbiased” way to carry out this sort of experiments is to extract the neighbours from disjoint sets having the *same* cardinality.

We experimented with this setup and divided our dataset into subsets of size 80533, 80533, 80534 – note that our pool of training neighbours has gotten even *smaller*!

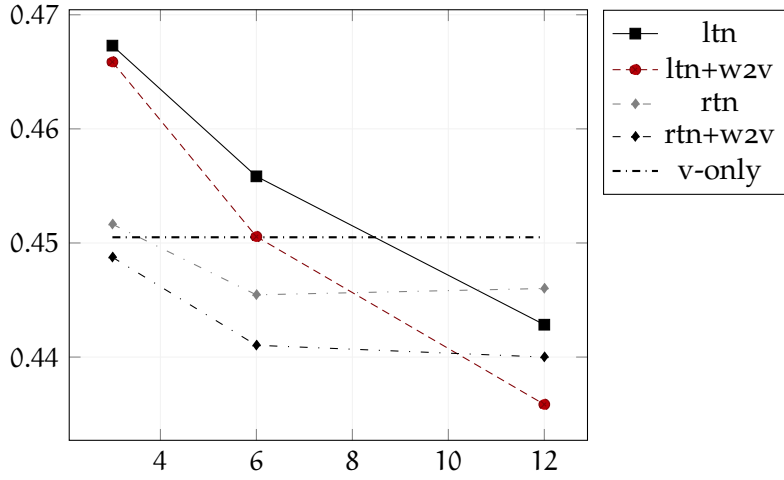
Some results with this setup are given in table 6.6, table 6.4, table 6.5.

We witness once again the same general relationships between models at $(m, M) = (3, 6)$ – LTN goes back to performing as expected at $(m, M) = (3, 6)$ – but again we see performance degradation as neighbourhoods expand.

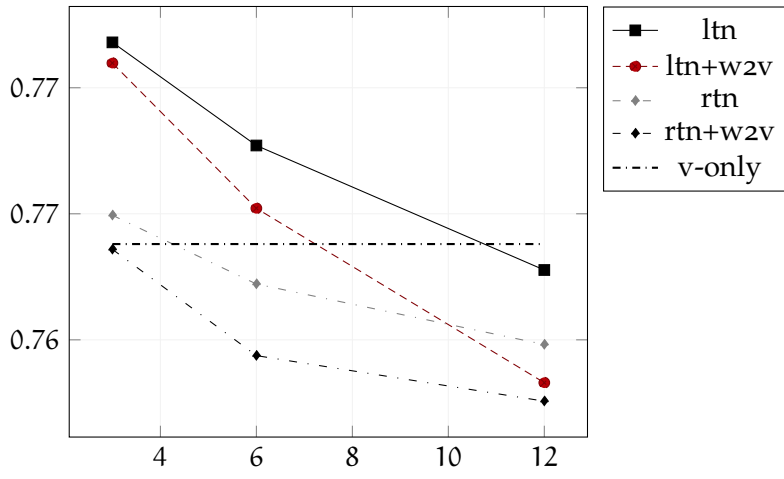
We can hypothesize several possible causes to explain the behaviour we see:

1. Even with a pool of 80k images to extract neighbours from at *test* time, neighbourhoods could be too noisy
2. A pool of 80k images to extract neighbours from at *training time* might be insufficient
3. When using neighbours from the whole dataset at all times there might be some degree of co-adaptation inflating the results.
4. The *asymmetry* in the size of neighbour pools in the setup of figure 6.7 causes performance degradation in itself.

These conjectures are non-exclusive.



(a) Mean mAP_{label} vs m



(b) Mean mAP_{img} vs m

Figure 6.2: Mean mAP_{img} and mAP_{label} vs m for visual neural architectures using test, training and validation sets closed under neighbourhood (figure 6.7)

We are not able to prove or disprove them, as at this time we are unable to run extensive experiments with more than 240k total images, however split.

Particularly, the third hypothesis could be disproved with 240K further *unique* images to build test neighbourhoods from.

Further studies need to be done to characterize the robustness of these models to changes in metadata vocabulary or noisiness, starting with studying their performance as a function of the size of the neighbour pools at training and test time and the IoU between them, particularly to ascertain where lays the “cutoff point”, as a function of neighbourhood pool size, after which larger neighbourhoods become detrimental due to noise.

Note, however, that there is no particular reason to extract neighbours from NUS-Wide or any other manually annotated dataset, since the ground truth is never used: very large Flickr datasets such as [TES⁺16] make excellent candidates. This is depicted in figure 6.8.

arch	neighbours	feed	mAP _{label}	mAP _{img}	rec _{label}	prec _{label}	rec _{img}	prec _{img}
ltn	id		* 46.73±0.21	* 77.68±0.20	41.24±1.01	43.27±1.68	72.08±0.20	51.89±0.18
ltn	w2v		46.59±0.10	77.60±0.14	41.01±1.04	43.46±1.06	72.06±0.13	51.86±0.14
rtn	id		45.16±0.22	76.99±0.23	40.54±1.69	44.24±1.81	71.51±0.15	51.48±0.17
rtn	w2v		44.88±0.22	76.86±0.23	39.98±1.86	43.91±1.47	71.26±0.18	51.29±0.18
ltn-vec	id		50.97±0.39	80.19±0.14	43.42±1.19	46.65±0.81	74.50±0.17	53.62±0.18
ltn-allvecs	id		51.86±0.27	79.99±0.17	44.93±1.39	46.13±1.38	74.38±0.15	53.49±0.16
lzip	id	id	59.13±0.32	82.44±0.27	52.28±1.54	48.65±2.49	76.72±0.21	55.01±0.22
lzip	id	w2v	61.47±0.32	82.59±0.27	53.86±1.51	51.05±2.12	76.90±0.27	55.02±0.22
ltwin	id	id	55.39±0.31	82.32±0.15	47.55±1.37	48.42±1.00	76.64±0.15	55.05±0.13
ltwin	id	w2v	63.17±0.33	* 83.77±0.11	54.10±1.23	52.62±1.28	78.09±0.08	55.81±0.14
ltwin	w2v	id	55.84±0.43	82.63±0.11	47.60±1.22	49.67±0.71	76.91±0.16	55.23±0.12
ltwin	w2v	w2v	* 63.24±0.19	83.71±0.11	54.61±1.37	51.52±1.41	78.05±0.12	55.77±0.18

Table 6.1: Summary of results for neighbourhood size $(m, M) = (3, 6)$ using test, training and validation sets closed under neighbourhood (figure 6.7)

arch	neighbours	feed	mAP _{label}	mAP _{img}	rec _{label}	prec _{label}	rec _{img}	prec _{img}
ltn	id		* 45.58±0.13	* 77.27±0.16	40.13±1.22	44.24±1.12	71.67±0.10	51.58±0.14
ltn	w2v		45.06±0.14	77.02±0.20	39.55±1.40	43.43±1.63	71.52±0.16	51.46±0.18
rtn	id		44.55±0.18	76.72±0.32	39.25±2.19	44.90±2.23	71.13±0.22	51.20±0.24
rtn	w2v		44.10±0.29	76.44±0.33	38.21±1.83	44.16±2.49	70.88±0.25	51.01±0.28
ltn-vec	id		49.73±0.24	79.86±0.12	43.17±1.44	45.60±2.01	74.16±0.10	53.38±0.14
ltn-allvecs	id		50.81±0.09	79.59±0.23	44.08±1.78	45.29±1.85	74.02±0.19	53.21±0.20
lzip	id	id	59.37±0.22	82.19±0.30	51.83±1.48	50.15±1.52	76.52±0.19	54.89±0.23
lzip	id	w2v	60.61±0.20	82.44±0.25	53.39±2.19	50.58±3.73	76.78±0.28	54.93±0.22
ltwin	id	id	53.58±0.16	81.90±0.13	46.34±1.31	47.83±1.17	76.16±0.12	54.72±0.14
ltwin	id	w2v	* 62.71±0.38	83.61±0.18	54.11±1.33	51.64±2.02	77.92±0.14	55.69±0.19
ltwin	w2v	id	54.12±0.27	82.21±0.13	46.17±1.41	47.92±1.66	76.48±0.15	54.92±0.18
ltwin	w2v	w2v	62.58±0.34	* 83.64±0.15	54.46±1.37	50.84±1.87	77.96±0.09	55.70±0.18

Table 6.2: Summary of results for neighbourhood size $(m, M) = (6, 12)$ using test, training and validation sets closed under neighbourhood (figure 6.7)

arch	neighbours	feed	mAP _{label}	mAP _{img}	rec _{label}	prec _{label}	rec _{img}	prec _{img}
ltn	id		44.28±0.11	★ 76.78±0.16	39.36±2.02	42.77±3.00	71.22±0.12	51.21±0.13
ltn	w2v		43.58±0.26	76.33±0.25	37.81±1.57	42.93±1.64	70.75±0.20	50.90±0.19
rtn	id		★ 44.60±0.23	76.48±0.33	38.36±1.81	44.70±2.20	70.88±0.22	51.04±0.29
rtn	w2v		44.00±0.34	76.26±0.32	36.90±2.26	45.51±2.71	70.76±0.21	50.95±0.18
ltn-vec	id		48.40±0.22	79.35±0.13	42.01±1.78	46.12±1.82	73.70±0.17	53.04±0.13
ltn-allvecs	id		49.66±0.21	79.23±0.22	43.09±1.47	45.71±1.57	73.63±0.14	52.94±0.16
lzip	id	id	58.31±0.24	81.18±0.36	50.70±3.20	50.45±2.59	75.63±0.35	54.00±0.32
lzip	id	w2v	59.45±0.58	81.99±0.52	52.44±1.45	48.61±2.71	76.53±0.21	54.79±0.27
ltwin	id	id	51.00±0.20	81.25±0.10	44.55±1.26	45.90±1.38	75.48±0.14	54.29±0.15
ltwin	id	w2v	61.88±0.23	83.41±0.15	53.73±0.98	50.65±1.67	77.80±0.05	55.59±0.16
ltwin	w2v	id	51.37±0.30	81.54±0.10	43.98±1.92	47.12±2.16	75.80±0.07	54.46±0.12
ltwin	w2v	w2v	61.91±0.24	★ 83.45±0.14	54.27±1.70	50.28±2.04	77.83±0.07	55.58±0.16
ltwin-2rnn	id	id	61.65±0.79	80.83±2.30	50.52±3.75	52.28±2.68	75.22±2.76	53.56±2.27
ltwin-2rnn	id	w2v	★ 61.99±0.70	82.83±0.38	51.75±3.09	50.12±2.55	77.42±0.23	55.35±0.22

Table 6.3: Summary of results for neighbourhood size $(m, M) = (12, 24)$ using test, training and validation sets closed under neighbourhood (figure 6.7)

arch	neighbours	feed	mAP _{label}	mAP _{img}	rec _{label}	prec _{label}	rec _{img}	prec _{img}
ltn	id		★ 46.69±0.24	★ 77.68±0.18	41.20±1.09	43.91±1.85	72.09±0.21	51.89±0.17
ltn	w2v		45.97±1.20	77.22±0.72	41.09±1.27	42.97±1.60	71.69±0.73	51.58±0.57
rtn	id		45.05±0.21	77.07±0.26	40.85±1.80	43.71±1.96	71.48±0.18	51.45±0.20
rtn	w2v		44.18±1.34	76.43±0.95	40.29±1.59	43.52±1.50	70.93±1.06	51.00±0.80
ltn-allvecs	id		51.67±0.19	79.96±0.21	44.45±1.47	46.20±1.26	74.29±0.12	53.43±0.16
lzip	id	id	59.17±0.30	82.37±0.23	51.69±2.57	49.75±3.39	76.66±0.17	54.96±0.23
lzip	id	w2v	61.23±0.21	82.52±0.20	54.79±2.70	51.13±2.93	76.85±0.13	54.97±0.20
ltwin	id	id	55.55±0.24	82.31±0.17	47.49±1.30	48.44±2.01	76.62±0.15	55.01±0.13
ltwin	id	w2v	★ 63.44±0.29	★ 83.74±0.18	54.28±1.24	52.31±1.01	78.08±0.13	55.79±0.18
ltwin-2rnn	id	id	60.32±0.74	81.61±0.44	52.66±1.83	47.26±3.25	76.24±0.43	54.54±0.34
ltwin-2rnn	id	w2v	57.36±6.59	80.55±3.05	49.50±3.93	45.30±4.88	75.29±2.95	53.83±2.10

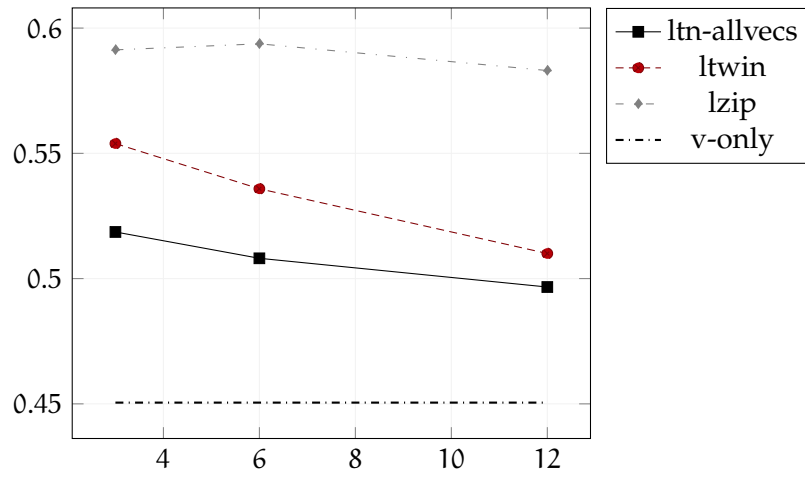
Table 6.4: Summary of results for neighbourhood size $(m, M) = (3, 6)$ using neighbours extracted from disjoint pools of $\frac{1}{3}$ the whole dataset at training, test and validation time

arch	neighbours	feed	mAP _{label}	mAP _{img}	rec _{label}	prec _{label}	rec _{img}	prec _{img}
ltn	id		* 45.52±0.14	* 77.24±0.25	40.92±1.58	42.98±1.23	71.73±0.21	51.60±0.20
ltn	w2v		44.84±0.86	76.71±0.74	39.71±1.70	42.29±0.76	71.14±0.75	51.19±0.58
rtn	id		44.58±0.27	76.74±0.30	39.59±2.17	43.81±2.41	71.19±0.18	51.24±0.27
rtn	w2v		43.41±1.34	76.04±1.02	38.68±1.42	43.30±1.40	70.49±1.03	50.70±0.81
ltn-allvecs	id		50.84±0.21	79.64±0.22	43.94±1.70	45.60±1.69	74.02±0.16	53.24±0.19
lzip	id	id	59.44±0.44	81.92±0.45	52.72±1.99	50.25±3.32	76.33±0.35	54.61±0.34
lzip	id	w2v	60.73±0.39	82.33±0.32	54.15±1.70	49.79±1.43	76.73±0.16	54.89±0.21
ltwin	id	id	53.50±0.16	81.86±0.10	46.43±1.64	47.65±1.54	76.13±0.15	54.70±0.14
ltwin	id	w2v	* 62.70±0.29	* 83.64±0.12	53.98±1.38	51.78±1.19	77.99±0.11	55.73±0.15
ltwin-2rnn	id	id	60.41±0.53	81.80±0.23	51.17±1.95	49.64±0.88	76.40±0.16	54.66±0.17
ltwin-2rnn	id	w2v	58.89±5.78	81.22±2.87	49.82±5.16	47.29±3.90	75.81±2.87	54.15±2.06

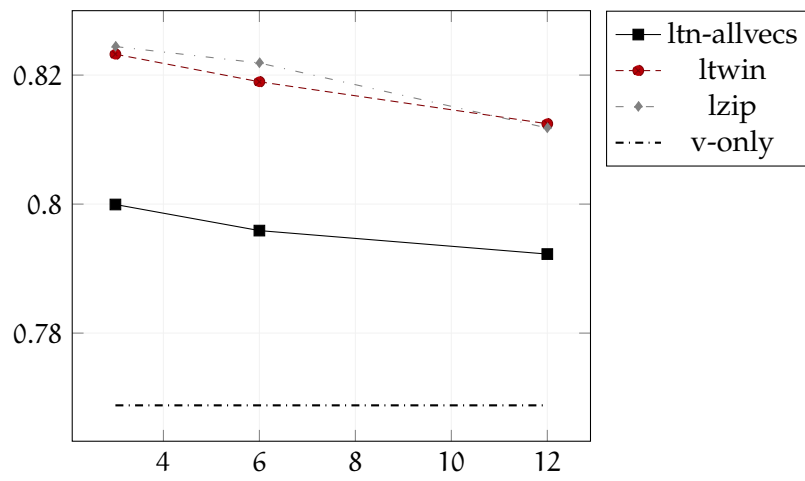
Table 6.5: Summary of results for neighbourhood size $(m, M) = (6, 12)$ using neighbours extracted from disjoint pools of $\frac{1}{3}$ the whole dataset at training, test and validation time

arch	neighbours	feed	mAP _{label}	mAP _{img}	rec _{label}	prec _{label}	rec _{img}	prec _{img}
ltn	id		44.18±0.25	76.65±0.22	39.15±2.17	42.40±2.66	71.11±0.16	51.17±0.14
ltn	w2v		43.29±0.66	76.03±0.69	37.77±1.38	42.47±1.04	70.48±0.58	50.71±0.48
rtn	id		* 44.72±0.20	* 76.66±0.28	38.37±2.38	44.98±3.06	71.09±0.21	51.18±0.19
rtn	w2v		43.26±1.19	75.92±0.99	37.13±1.93	44.35±1.82	70.38±0.97	50.64±0.71
ltn-allvecs	id		49.68±0.17	79.16±0.17	42.92±1.84	45.35±2.16	73.57±0.13	52.89±0.13
lzip	id	id	58.38±0.60	81.18±1.74	49.10±2.36	53.14±3.84	75.35±2.05	53.93±1.56
lzip	id	w2v	59.07±0.33	81.96±0.40	52.42±2.07	48.10±3.88	76.41±0.28	54.64±0.33
ltwin	id	id	50.92±0.18	81.26±0.12	44.38±1.66	46.33±2.02	75.53±0.12	54.28±0.13
ltwin	id	w2v	* 61.75±0.20	* 83.35±0.15	53.97±1.50	50.36±2.24	77.71±0.08	55.53±0.18
ltwin-2rnn	id	id	61.10±0.79	81.78±0.50	51.61±1.65	49.25±3.17	76.50±0.37	54.63±0.40
ltwin-2rnn	id	w2v	59.37±4.31	81.42±2.45	50.40±4.03	46.68±3.48	76.16±2.19	54.41±1.55

Table 6.6: Summary of results for neighbourhood size $(m, M) = (12, 24)$ using neighbours extracted from disjoint pools of $\frac{1}{3}$ the whole dataset at training, test and validation time

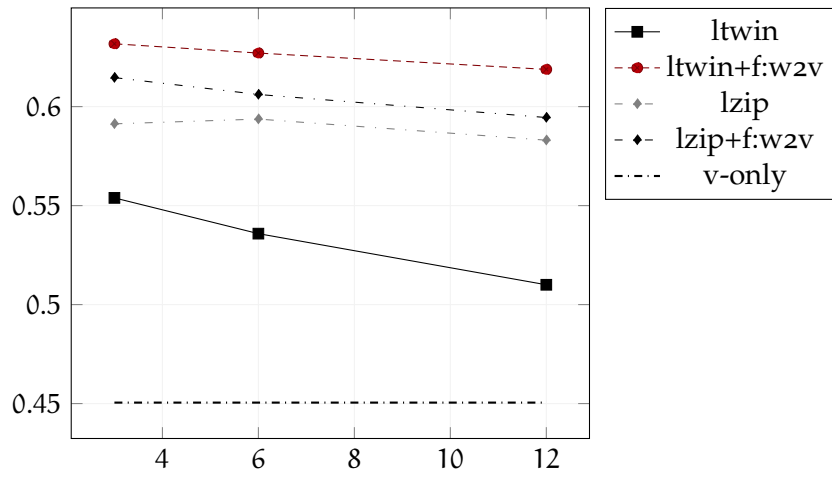


(a) Mean mAP_{label} vs m

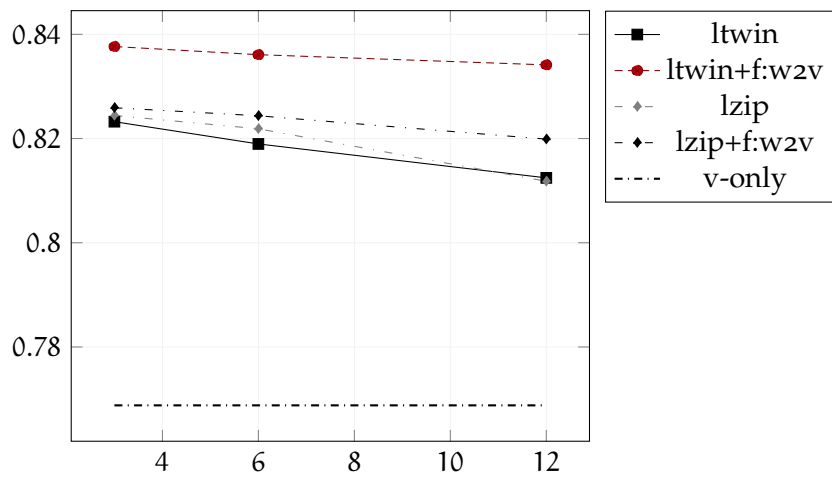


(b) Mean mAP_{img} vs m

Figure 6.3: Mean mAP_{img} and mAP_{label} vs m for select joint models under with $\pi = \text{id}$ using test, training and validation sets closed under neighbourhood (figure 6.7)



(a) Mean mAP_{label} vs m



(b) Mean mAP_{img} vs m

Figure 6.4: Mean mAP_{img} and mAP_{label} vs m for select joint models, various choices of π using test, training and validation sets closed under neighbourhood (figure 6.7)

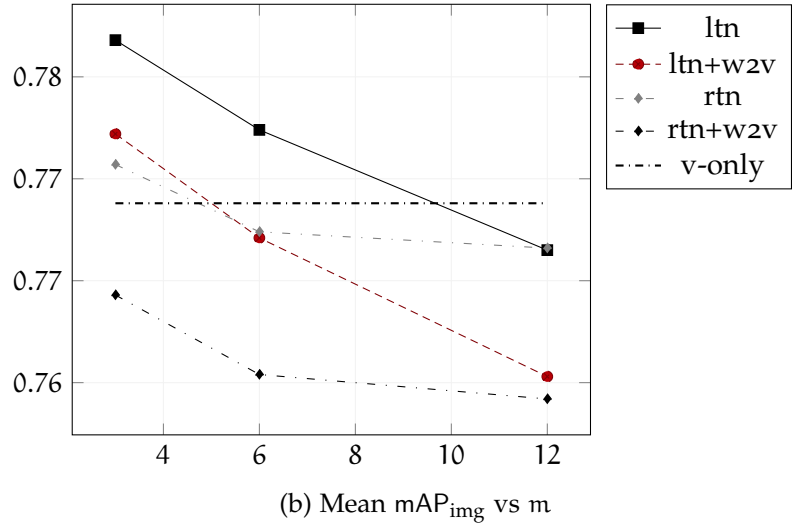
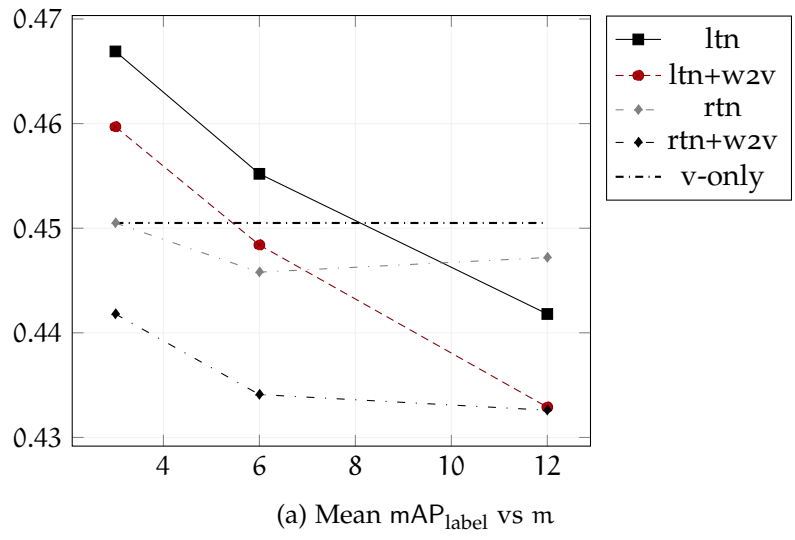


Figure 6.5: Mean mAP_{img} and mAP_{label} vs m for visual neural architectures using neighbours extracted from disjoint pools of $\frac{1}{3}$ the whole dataset at training, test and validation time

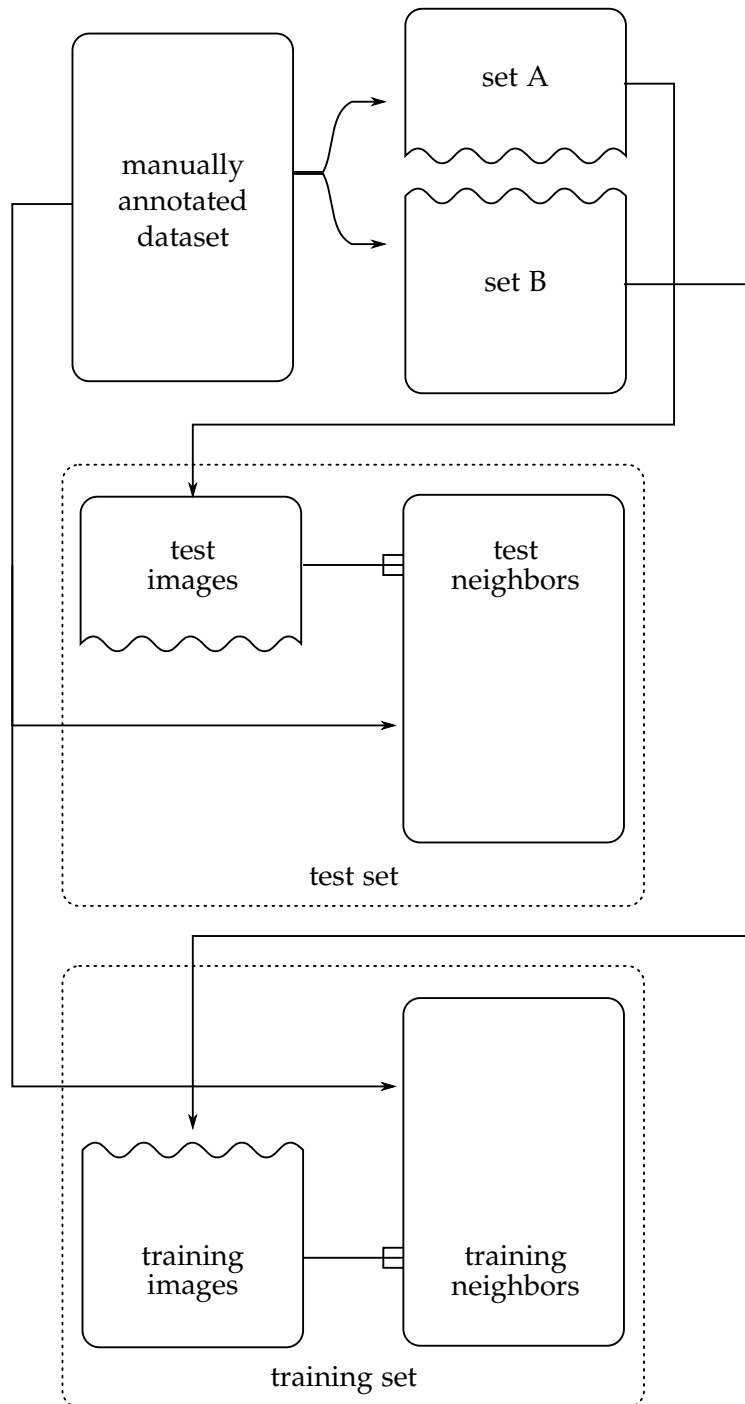


Figure 6.6: Neighbourhood generation from entire dataset

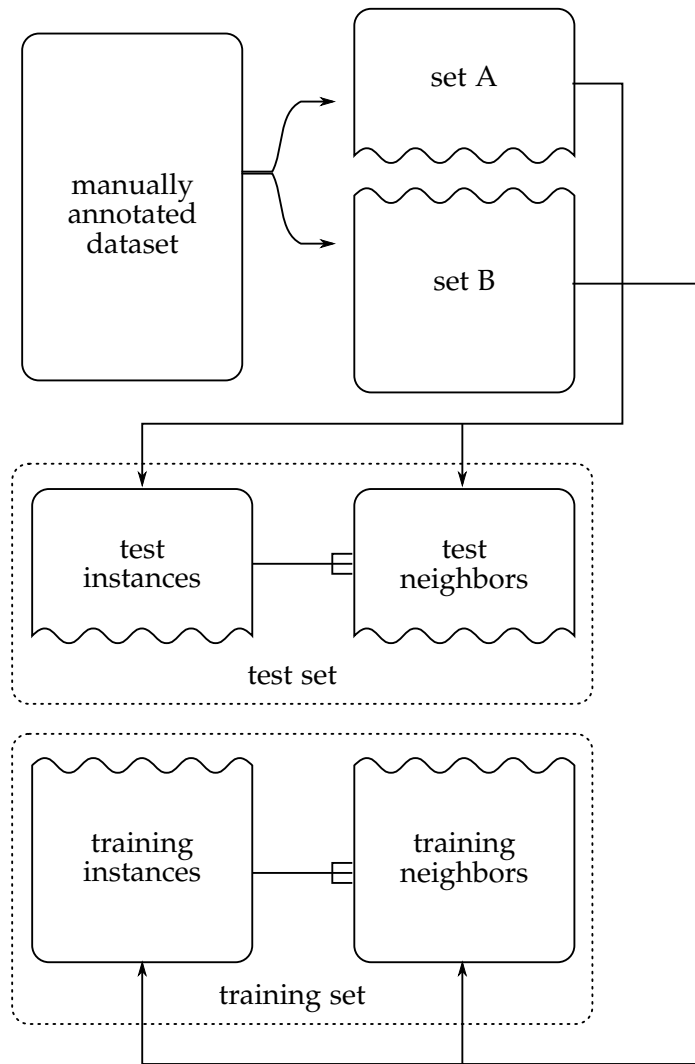


Figure 6.7: Training, test and validation sets closed under neighbourhood

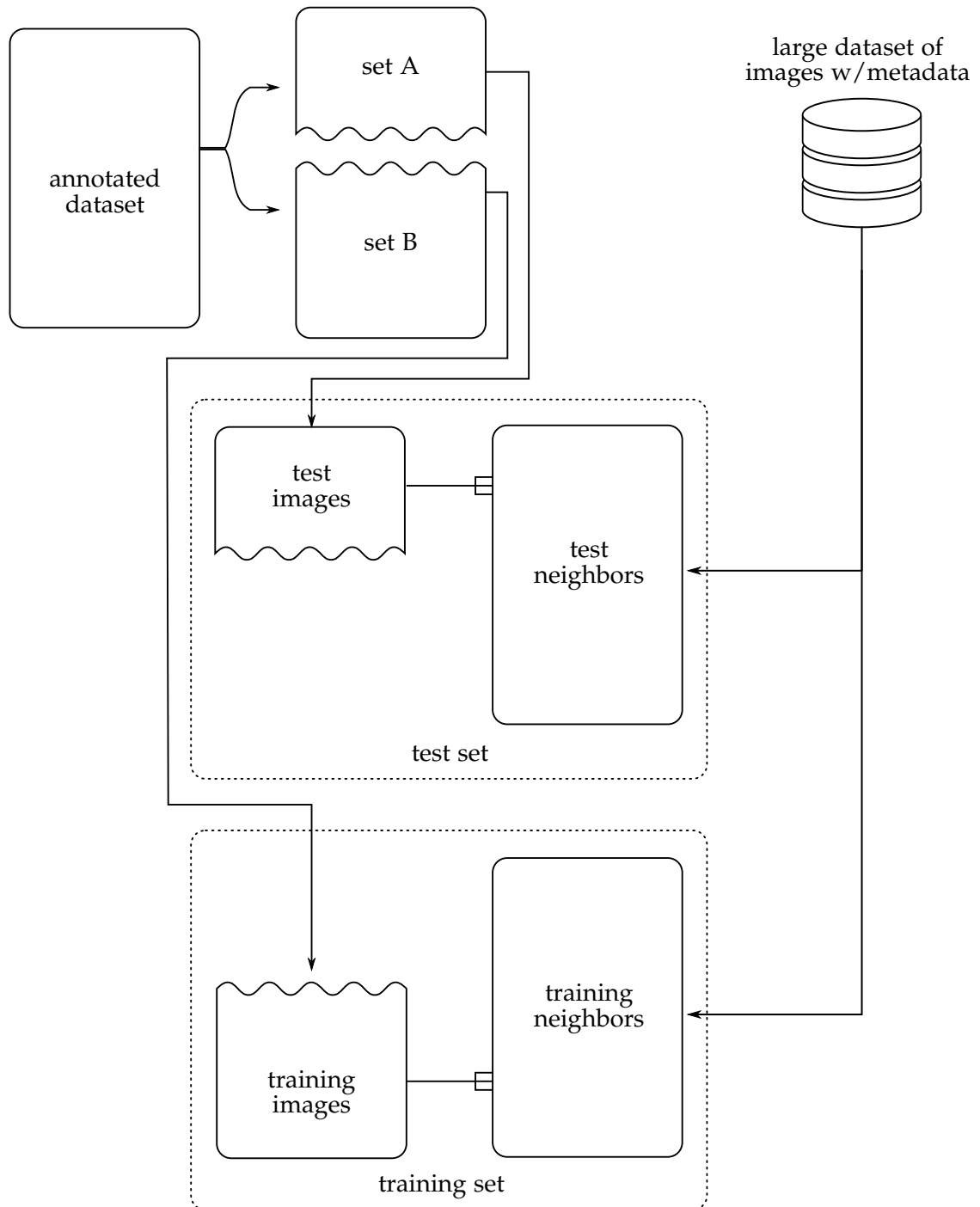


Figure 6.8: Proposed procedure for neighbourhood generation

Chapter 7

Conclusions

We have seen that the baseline of the visual model based on metadata neighbours of [JBFF15] can be improved within the same framework with semantic mappings and architectural modifications involving RNNs.

Moreover, we have described a framework that extends upon that of [JBFF15] and characterized the performance of variety of joint models built around it: we have seen that the performance of different neural architectures exhibits significant variability.

We have seen that semantic mappings can be highly effective in improving performance, besides achieving robustness to changes in metadata vocabulary and quality of neighborhoods.

One interesting finding is that the effect of using semantic mappings in neighbourhood generation seems to be overshadowed by their effect on the data input into the neural architecture, suggesting that it might not always be worth the effort to incorporate them in the NN mechanism.

We have also shown that imputing semantic dictionaries through a knowledge transfer approach can be helpful when the intersection between an existing semantic dictionary and the tag vocabulary is small; we believe this can be further investigated as a useful technique to improve the performance of classifiers that involve such mappings.

We have seen that RNN-based models can perform very well, although the results of chapter 6 suggest that their effect with excessively small or noisy neighbour pools might be detrimental and needs further investigation to be characterized precisely.

We have seen that, in general, there is far potential to attack “hard” classes through metadata than there is to improve per-image performance; this might make the models presented here more relevant in those contexts, particularly when dealing with imbalanced datasets and when there is a high cost for neglecting some classes.

Algorithmic bias, which has recently become a matter of concern in the public eye, is an immediate consequence of the above-mentioned issues: we expect its relevance not to decline anytime soon.

Finally we have proposed a new, unbiased experimental protocol to carry out experiments with this sort of models and to measure the robustness of models to changing metadata vocabulary and neighborhood quality.

This page intentionally left blank.

Acknowledgements

Very special thanks go to my supervisor, Lamberto Ballan, who went above and beyond the call of duty to remove obstacles in my path, while letting me walk it on my own two feet.

The leaves and trees of chapter 1 are from my mother's sketchbook; she also provided support and fine food.

A huge thank you also goes to my friends and occasional proofreaders, particularly Nicola, Pippo, Theo, Alessandro and Elena.

Finally, a shout out goes to the whole of the Visual Intelligence and Machine Perception research group at UniPD for the stimulating discussions and interesting ideas I've been exposed to.

This page intentionally left blank.

Bibliography

- [BDD⁺12] R. Bauer, G. D'Angelo, D. Delling, A. Schumm, and D. Wagner. The Shortcut Problem - Complexity and Algorithms. *Journal of Graph Algorithms and Applications*, 16(2):447–481, 2012. doi:10.7155/jgaa.00270.
- [Cab17] S. Cabello. Subquadratic Algorithms for the Diameter and the Sum of Pairwise Distances in Planar Graphs. *arXiv:1702.07815 [cs]*, Feb. 2017. URL <http://arxiv.org/abs/1702.07815>. arXiv: 1702.07815.
- [cs219] CS231n Convolutional Neural Networks for Visual Recognition, May 2019. URL <http://web.archive.org/web/20190514151219/cs231n.github.io/>.
- [CTH⁺09] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng. NUS-WIDE: a real-world web image database from National University of Singapore. In *Proceeding of the ACM International Conference on Image and Video Retrieval - CIVR '09*, page 1, Santorini, Fira, Greece, 2009. ACM Press. doi:10.1145/1646396.1646452.
- [CVMG⁺14] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- [DDS⁺09] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
- [DS02] B. DasGupta and G. Schnitger. On the Computational Power of Analog Neural Networks. In *The Handbook of Brain Theory and Neural Networks*, page 14. 2002.
- [EVGW⁺10] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010. doi:10.1007/s11263-009-0275-4.
- [FP12] D. Forsyth and J. Ponce. *Computer vision: a modern approach*. Pearson, Boston, 2nd ed edition, 2012.
- [GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [GHF12] Gang Wang, D. Hoiem, and D. Forsyth. Learning Image Similarity from Flickr Groups Using Fast Kernel Machines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2177–2188, Nov. 2012. doi:10.1109/TPAMI.2012.29.

- [GMVS09] M. Guillaumin, T. Mensink, J. Verbeek, and C. Schmid. TagProp: Discriminative metric learning in nearest neighbor models for image auto-annotation. In *2009 IEEE 12th International Conference on Computer Vision*, pages 309–316, Kyoto, Sept. 2009. IEEE. doi:10.1109/ICCV.2009.5459266.
- [GVS10] M. Guillaumin, J. Verbeek, and C. Schmid. Multimodal semi-supervised learning for image classification. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 902–909, San Francisco, CA, USA, June 2010. IEEE. doi:10.1109/CVPR.2010.5540120.
- [Har15] A. W. Harley. An interactive node-link visualization of convolutional neural networks. In *ISVC*, pages 867–877, 2015.
- [Hin] G. Hinton. Neural Networks for Machine Learning. URL <http://web.archive.org/web/20161114232611/https://www.coursera.org/learn/neural-networks>.
- [Hin86] G. E. Hinton. Distributed representations. Technical report, Carnegie Mellon University, 1986.
- [HZD⁺16] H. Hu, G.-T. Zhou, Z. Deng, Z. Liao, and G. Mori. Learning Structured Inference Neural Networks with Label Relations. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2960–2968, Las Vegas, NV, USA, June 2016. IEEE. doi:10.1109/CVPR.2016.323.
- [HZRS15] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv:1502.01852 [cs]*, Feb. 2015. URL <http://arxiv.org/abs/1502.01852>. arXiv: 1502.01852.
- [HZRS16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [JBFF15] J. Johnson, L. Ballan, and L. Fei-Fei. Love thy neighbors: Image annotation by exploiting image metadata. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4624–4632, 2015.
- [KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [LBB⁺98] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.
- [LBD⁺89] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
- [Lec89] Y. Lecun. Generalization and network design strategies. In *Connectionism in perspective*. Elsevier, 1989.

- [LUB⁺16] X. Li, T. Uricchio, L. Ballan, M. Bertini, C. G. M. Snoek, and A. D. Bimbo. Socializing the Semantic Gap: A Comparative Survey on Image Tag Assignment, Refinement, and Retrieval. *ACM Computing Surveys*, 49(1):1–39, June 2016. doi:10.1145/2906152.
- [LXH⁺17] F. Liu, T. Xiang, T. M. Hospedales, W. Yang, and C. Sun. Semantic Regularisation for Recurrent Image Annotation. pages 4160–4168. IEEE, July 2017. doi:10.1109/CVPR.2017.443.
- [MCCD13] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*, Jan. 2013. URL <http://arxiv.org/abs/1301.3781>. arXiv: 1301.3781.
- [Mil95] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.
- [Mit97] T. Mitchell. *Machine Learning*. McGraw-Hill international editions - computer science series. McGraw-Hill Education, 1997. URL <https://books.google.it/books?id=x0GAngEACAAJ>.
- [ML12] J. McAuley and J. Leskovec. Image Labeling on a Network: Using Social-Network Metadata for Image Classification. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, editors, *Computer Vision – ECCV 2012*, volume 7575, pages 828–841. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-33765-9-59.
- [MP72] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. Mit Press, 1972. URL <https://books.google.it/books?id=0w10AQAAIAAJ>.
- [MRS08] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.
- [NG19] NLX-Group. Obtaining word embeddings from a WordNet ontology. Contribute to nlx-group/WordNetEmbeddings development by creating an account on GitHub, May 2019. URL <https://github.com/nlx-group/WordNetEmbeddings>. original-date: 2018-05-25T10:39:08Z.
- [NHH15] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528, 2015.
- [oed] Context. in *The Oxford English Living Dictionary*. <http://web.archive.org/web/20190323145341/https://en.oxforddictionaries.com/definition/context>. Accessed: 2019-03-21.
- [PMB13] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

- [Pow] V. Powell. Image Kernels explained visually. URL <http://web.archive.org/web/20190329125343/http://setosa.io/ev/image-kernels/>.
- [RN16] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Always learning. Pearson, 2016. URL <https://books.google.it/books?id=XS9CjwEACAAJ>.
- [Roj96] R. Rojas. *Neural Networks: A Systematic Introduction*. Springer-Verlag, Berlin, Heidelberg, 1996.
- [SBRS] C. Saedi, A. Branco, J. A. Rodrigues, and J. Silva. WordNet Embeddings. In *Proceedings of the 3rd Workshop on Representation Learning for NLP*, pages 122–131.
- [SHK⁺14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL <http://jmlr.org/papers/v15/srivastava14a.html>.
- [son19] Example of 2d Convolution, Jan. 2019. URL http://web.archive.org/web/20190107213933/http://www.songho.ca/dsp/convolution/convolution2d_example.html.
- [SS95] H. T. Siegelmann and E. D. Sontag. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150, 1995.
- [SWS⁺01] A. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22:1349–1380, Jan. 2001. doi:10.1109/34.895972.
- [SZ15] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.
- [TES⁺16] B. Thomee, B. Elizalde, D. A. Shamma, K. Ni, G. Friedland, D. Poland, D. Borth, and L.-J. Li. YFCC100m: the new data in multimedia research. *Communications of the ACM*, 59(2):64–73, Jan. 2016. doi:10.1145/2812802.
- [w2v19] word2vec at Google Code Archive - Long-term storage for Google Code Project Hosting., May 2019. URL <http://web.archive.org/web/20190501011336/https://code.google.com/archive/p/word2vec/>.
- [WM97] D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr. 1997. doi:10.1109/4235.585893.
- [ZWZ⁺19] J. Zhang, Q. Wu, J. Zhang, C. Shen, and J. Lu. Mind your neighbours: Image annotation with metadata neighbourhood graph co-attention networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR’19)*, 2019. URL http://openaccess.thecvf.com/content_CVPR_2019/papers/Zhang_Mind_Your_Neighbours_Image_Annotation_With_Metadata_Neighbourhood_Graph_Co-Attention_CVPR_2019_paper.pdf.