Feature analysis of two stream convolutional neural networks

Will Price

A thesis presented for the degree of Bachelor of Science

University of BRISTOL

Department of Computer Science

May 2017

Abstract

We investigate visualisation methods to provide insight into the features learnt by a CNN. We perform a survey of existing visualisation methods categorising them into broad categories with a particular focus on methods for attention mapping; methods producing heatmaps for specific network inputs indicating the relevance of regions across the input to the activation of a given neuron.

We apply Excitation Backprop (EBP)[54], an attention mapping method developed and tested on object detection networks, to the two stream CNN (2SCNN)[41], a network architecture designed for action recognition. The network is composed of two separate network towers: the spatial tower, receiving a single video frame as input; and the temporal tower, operating over a stack of optical flow frames derived from the video sequence.

We generate attention maps using two variations of EBP: contrastive and non-contrastive for two pretrained models of the 2SCNN. Of the two models, one is trained on UCF101[42], an action recognition dataset sourced from YouTube; the other on BEOID[5], an egocentric object-interaction action recognition dataset We extend EBP on the temporal stream to produce attention maps on a per frame basis. We use a sliding window approach to produce a sequence of attention maps from which we can determine attention changes after sliding the temporal window. We show that temporal attention maps highlight salient regions over a temporal window determined by the network input. We analyse the attention maps generated for the spatial stream confirming that the network uses environmental cues for action prediction.

We analyse the learnt features in the network, and study the attention maps over a sequence of frames. We assess the *jitter* in attention map sequences, and compare the attention location of the network to the action location estimated by operator gaze on BEOID. We also visualise the first and second layer filters from the network.

The thesis concludes with a summary of contributions and a direction into future work.

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of Bachelor of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Will Price, May 2017.

Acknowledgements

This work could not have been completed without the tremendous help and support of my supervisor, Dima Damen. I look forward to continuing to better understand neural networks under her supervision over the next 4 years.

I would also like to thank my parents for their continued support and encouragement they have given me throughout my life, affording me many opportunities that have helped me reach where I am today.

Contents

List of Tables

List of Figures

1 Introduction

Action Recognition in Computer Vision refers to approaches that aim to infer the action or actions of an actor or actors using visual observations in the form of images or videos. In this thesis we further constrain the definition to only inferring actions from video sequences (sequences of images captured by video cameras at regular intervals). Action recognition from video has many critical applications[32] such as detecting suspicious behaviours of travellers in airports from CCTV footage; recognising the fall of an elderly person who lives alone; and ensuring the safety of the operator of a machine by automatically stopping the machine in case of an accident.

Convolutional Neural Networks (CNNs), a form of supervised deep learning model, have recently been used to obtain state of the art results in object detection in images[20]. Naturally researchers have questioned whether these performance increases translate to action recognition. The results of extending CNN architectures to cope with video sequences have yielded similar increases in performance [10], [47].

One downside of CNNs compared to other models such as decision trees is their lack of transparency in predictions: why does the model classify this particular video sequence of a person performing pull ups as skipping? With a decision tree we would be able to provide an explanation based on the features from the root of the tree to the branch, but there's no obvious analogous technique to understand a CNN's prediction.

There are efforts made to develop techniques to help understand the features learnt by CNNs to aid debugging, but most of these have been developed and tested on object detection networks; there is little research to see whether the techniques generalise to networks trained for other tasks such as action recognition.

This thesis surveys visualisation techniques available to understand CNNs. A method called Excitation Backpropagation (EBP)[54] is used and extended to produce attention maps (heatmaps indicating the regions of importance in activating a given neuron in the network) across sequences of frames from a video sequence classified by a 2SCNN[40] trained for action recognition[1].

Contribution summary:

- Survey of visualisation methods for CNNs organised into hierarchy.
- Visual analysis of the first and second layer filters learnt by each stream.
- Extension of EBP for use on the temporal network streams to generate attention maps on a per frame basis.
- A quantitative assessment of attention map sequences for *jitter* demonstrating the inferiority of contrastive EBP on the spatial network for UCF101.
- A quantitative assessment of attention map quality on the BEOID dataset by comparing the distance between action location (using gaze location as a proxy) and attention map peak location.

[1] There are other architectures for action recognition, but they are out of the scope of this investigation

- A qualitative assessment of successful and pathological attention map sequences for both datasets.

2 Background

We introduce the basic concepts of artificial neural networks in sec. 2.1 and convolutional neural networks in sec. 2.2. We then survey visualisation techniques developed to understand the different aspects of trained CNNs in sec. 3 with a particular focus on excitation back propagation in sec. 4.

2.1 Artificial neural networks (ANNs)

Biology is a rich source of inspiration for techniques in computer science. Artificial neural networks (ANNs) form a strand of biologically inspired computational models based on the learning and communication processes in the brain. To understand neural networks, we will examine each concept from the bottom up step by step until we arrive at the modern model of an artificial neural network. First we shall examine *artificial neurons*, of which there are several models, the earliest being the McCulloch-Pitts neuron[26], followed by the perceptron[34]. We will then demonstrate how one can form a network made up of artificial neurons using perceptrons. We then briefly discuss the computational challenges scaling these networks up to process images or videos leading into an introduction to convolutional neural networks, a constrained form ANN architecture making certain assumptions improving their computational tractability.

2.1.1 The McCulloch-Pitt's neuron

The McCulloch-Pitt's neuron is mostly a historical curiosity, and if the evolution of artificial neural networks doesn't interest you skip ahead to the perceptron.

Warren McCulloch and Walter Pitts were arguably the first to provide a mathematical model of neuron inspired by biology, they developed a logical calculus describing neuron behaviour[26]. Their model neuron, known as McCulloch-Pitt's neuron (MCP), was shown to be computationally universal; every network of MCP neurons encoded an equivalent logical proposition.

MCP neurons have a set of inputs, the sum of which is compared to a threshold which determines whether the neuron fires or not. Both excitatory and inhibitory signals were modelled, if an incoming inhibitory connection is firing, then the output is completely inhibited regardless of the firing of the other incoming signals. Cells produce a binary output, they either fire or not, there is no notion of firing strength.

2.1.2 The Perceptron

The next major contribution to the realisation of ANNs following McCulloch and Pitt's work was the Perceptron[34] developed by Frank Rosenblatt, initially conceived as a

physical device for learning to recognise patterns in images or sounds (each of these using the same principles, but with different inputs), it was later formulated as an algorithm for implementation in software.

The perceptron, in modern machine learning terms, is a supervised learning algorithm that produces a binary linear classifier. First we'll step through each term in this definition before presenting the perceptron:

- *learning* is concerned with the problem of constructing a function $f : X \to Y$, where X denotes the feature space, Y denotes the label space.
- *feature space*, the vector space of data from predict a class (domain of f). The data available to be used as input to the model influences the choice of domain, e.g. an image of size $W \times H$ could be analysed for edges and those used as the feature vector, or every pixel could be used in $W \times H$ long vector as input to the predictive function, amongst many other possibilities.
- *label space*, the vector space in which the desired result resides (co-domain of f).
- *supervised learning* uses a labelled training set $X_{\text{train}} = \{(\boldsymbol{x}_0, y_0), \ldots, (\boldsymbol{x}_n, y_n)\}$ to learn f where X_{train} is used as a set of examples by the learning algorithm used to construct f.
- *classification* further refines the definition of the function f to be learnt. Classification is about learning a function that predicts one of a finite number of labels hence the label space will be a finite set of labels/classes.
- *binary classification* specifies that the *label space* consists of a set of 2 labels/classes, usually referred to as the *positive* and *negative* classes.
- a *linear classifier* implies that the learnt model is of the form $\boldsymbol{w} \cdot \boldsymbol{x} > 0$ where $\boldsymbol{w}$ is a vector of weights and $\boldsymbol{x}$ is the instance in feature space to be classified. The classifier predicts the positive class if $\boldsymbol{w} \cdot \boldsymbol{x} > 0$, and the negative class if $\boldsymbol{w} \cdot \boldsymbol{x} < 0$. If $\boldsymbol{w} \cdot \boldsymbol{x} = 0$ then the instance lies on the decision boundary and we have to make a decision as to which class to predict (e.g. randomly choose positive class 50% of the time).

A graphical representation of the perceptron is given in fig. 2.1, each element x_i of the feature vector forms an input node on a graph, elements of the weight vector w_i form edges from the corresponding input (x_i) to the perceptron body which computes the weighted sum of all the inputs. An additional input $x_0 = 1$ known as the perceptron *bias* is implicitly added to the input determining the intersection of the linear boundary with the y-axis. One can think of inputs flowing along the edges into the perceptron body, as they flow along the edge they are multiplied by the edge's weight, finally the perceptron body sums its inputs producing the perceptron output $z = \boldsymbol{w} \cdot \boldsymbol{x}$.

The perceptron learning algorithm constructs $\boldsymbol{w}$ from a set of labelled training examples $\mathcal{X} = \{(\boldsymbol{x}_0, y_0), \ldots, (\boldsymbol{x}_n, y_n)\}$ where $\boldsymbol{x}_i$ is the feature representation of the i-th training example, and y_i is the true label of the example (a numerical encoding of its class, typically 1 represents the positive class, and -1 the negative class).

Algorithm 1 learns a $\boldsymbol{w}$ such that the resulting linear classifier correctly classifies all

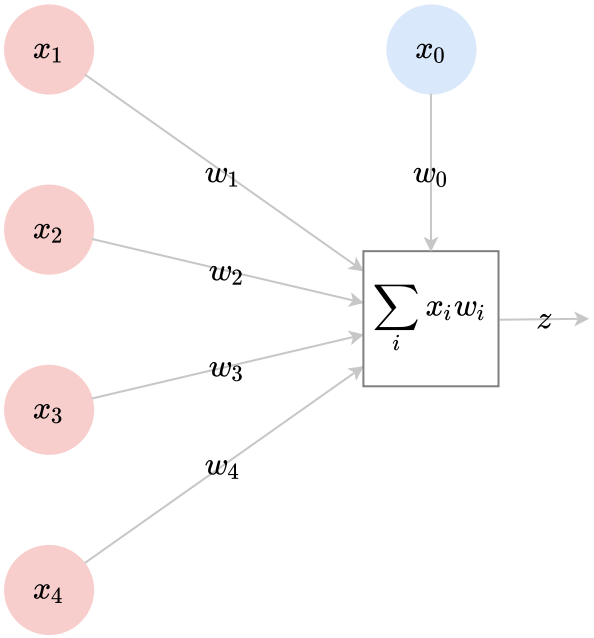


Figure 2.1: Graphical representation of the perceptron

examples on the training set (if possible, otherwise the algorithm fails to terminate).

Data: Training set: $\mathcal{X} = \{(\boldsymbol{x_0}, y_0), \cdots, (\boldsymbol{x_n}, y_n)\}$
Result: Binary classifier: $\boldsymbol{w} \cdot \boldsymbol{x} > 0$
$\boldsymbol{w} \leftarrow \boldsymbol{0}$;
converged $\leftarrow$ false;
while converged $=$ *false* **do**
  converged $\leftarrow$ true;
  for $i = 1$ *to* $|\mathcal{X}|$ **do**
    if $y_i \boldsymbol{w} \cdot \boldsymbol{x_i} \leq 0$ **then**
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta y_i \boldsymbol{x_i}$;
      converged $\leftarrow$ false;
    end
  end
end

Algorithm 1: Perceptron training

The idea is to iteratively build up a weight vector $\boldsymbol{w}$ that correctly classifies all training data. Initially starting with the zero vector will result in the misclassification of all training examples as they will all lie on the decision boundary, $\boldsymbol{w} \cdot \boldsymbol{x} = 0$. The core of the algorithm depends on interpreting the dot product as a measure of similarity: the dot product produces a more positive result if $\boldsymbol{w}$ and $\boldsymbol{x}$ are similar, and a negative result if $\boldsymbol{w}$ and $\boldsymbol{x}$ are dissimilar[1] . By adding weighted training feature vectors and factoring in the correct sign of y_i, $\eta \boldsymbol{x}_i y_i$ to the weight vector $\boldsymbol{w}$, we increase the similarity of the new weight vector with the training example resulting in a more positive dot product between $\boldsymbol{w}$ and $\boldsymbol{x}_i y_i$ which is more likely to pass the decision threshold.

[1]Consider the definition of the dot product: $\boldsymbol{a} \cdot \boldsymbol{b} = |\boldsymbol{a}||\boldsymbol{b}| \cos(\theta)$, when $\boldsymbol{a}$ and $\boldsymbol{b}$ are orthogonal, $\theta = 90°$ hence $\boldsymbol{a} \cdot \boldsymbol{b} = 0$, when the vectors point in the same direction $\theta = 0°$ then $\boldsymbol{a} \cdot \boldsymbol{b} = |\boldsymbol{a}||\boldsymbol{b}|$. Finally, when the vectors point in the opposite direction $\theta = 180°$ hence $\boldsymbol{a} \cdot \boldsymbol{b} = -|\boldsymbol{a}||\boldsymbol{b}|$

The perceptron learns a linear classifier, which is only of use if the data is linearly separable, if it isn't then we have to consider alternatives. Minsky and Papert used the following example (fig. 2.2) in Perceptrons[27] to demonstrate the limitations of a single perceptron, the figure shows the function XOR where the red points (negative examples) cannot be separated from the green points (positive examples) with a linear boundary, so a boundary like that shown in fig. 2.3 is needed to separate the data.

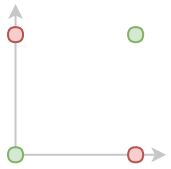


Figure 2.2: Perceptrons fail to learn the XOR function due to the necessity of a non-linear decision boundary

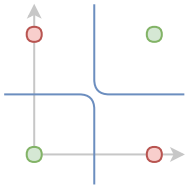


Figure 2.3: XOR with non linear decision boundary

To solve the XOR problem we can construct individual perceptrons that simulate Boolean functions and then use the XOR propositional decomposition ($p \oplus q = (p \vee q) \wedge \neg(p \wedge q)$) to construct a network that implements XOR, but this solution negates the main benefit of using a learning algorithm in the first place: we want the machine to form a solution.

Perceptrons are capable of learning non linear decision boundaries with one of two modifications. The first technique is to replace the dot product with a *kernel*, a function with properties similar to that of the dot product. Use of a kernel as a replacement for the dot product can be thought of as a transformation of the instances into a new space in which a linear decision boundary is learnt. A kernel is chosen such that it is likely that the data will be linearly separable in this new space. Alternatively, the other technique is to stack perceptrons so that the output of one is used as (one of) the input(s) to another perceptron, the resulting network of perceptrons is called a *multilayer perceptron (MLP)*.

When using MLPs we have to adapt the perceptron's output to be followed by a non-linear transformation ϕ; the reason for this is that if we otherwise stack perceptrons without modification the network would compute a combination of linear transformations and any combination of linear transformations can be represented by a single linear transformation[2] i.e. MLPs without non linearity applied to the output of each unit are no more expressive than a single perceptron; the complexity of the decision boundaries learnt by MLPs is due to the successive application of linear transformations and non linearities.

A small multilayer perceptron network is given in fig. 2.4. Each circle represents the body of a perceptron in which the weighted sum of its inputs are calculated and then passed through an activation function ϕ. Perceptrons are labelled using the notation $a_j^{(l)}$ where l indicates the zero-based index of the layer the perceptron lies within, and j the index

[2] A perceptron unit computes an affine transform. Affine transforms can be combined into a new affine transform representing the transform computed by combining the constituent transforms.

of the perceptron within the layer, e.g. $a_0^{(1)}$ is the first (index 0) perceptron in layer 2 (index 1). Each edge between perceptrons indicates the connectivity and weight between them. For example, $a_0^{(1)}$ has two incoming connections: one from $a_0^{(0)}$ with a weight +1 and another from $a_1^{(0)}$ with a weight 0, it will output the value $\phi\left(1 \cdot a_0^{(0)} + 0 \cdot a_1^{(0)}\right)$

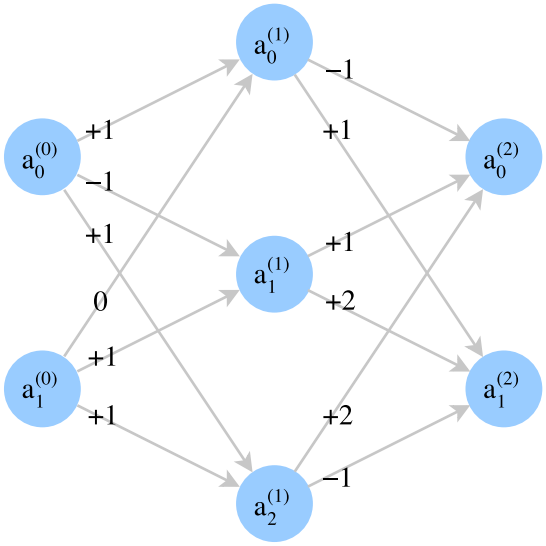


Figure 2.4: Example multilayer perceptron with a single hidden layer

A forward pass of the network in fig. 2.4 is computed using the activation function $\phi(x) = \max(x, 0)$ in fig. 2.5. We traverse the graph from left to right, computing the values of every perceptron in each layer before moving to the next layer. The edges are relabelled with the product of the weight and input to the edge, the diamonds below each perceptron show the sum of the weighted inputs (the sum of the values labelled on the edges) and the diamonds above show the output value of the perceptron after passing the weighted sum of inputs through the activation function ϕ.

Combining multiple perceptrons into a network forming a multilayer perceptron brings us closer to the modern artificial neural network, however we now have a new problem: learning the weights of all the perceptrons. Since the weight vectors of the perceptrons in the network are not independent, changing one will effect inputs deeper in the network causing a change in the final output, meaning we cannot use Algorithm 1. An exhaustive search over the weights of the perceptrons would be able to find an optimal weight configuration, but would be computationally intractable due to the combinatorial nature of the search. Training multilayer perceptrons was the main impediment to their use until the process of *error back propagation* (first developed by Kelley[19] and Bryson[7][37]) was applied to the problem by Paul Werbos[37]. Back propagation gives us a tool to understand how modifying each component of the weight vector of each perceptron changes the output of the network. A loss function is defined that specifies how the output of the network differs from the desired output. The partial derivatives of the loss function with respect to each weight component $\frac{\partial a_i^{(n)}}{\partial w_j^{(l)}}$ are calculated through use of the chain rule. Having obtained the partial derivatives of the loss function with respect to the weights of the network, we can perform gradient descent in the weight space to tweak the weights in such a way that the loss function is minimized thus causing the output of the network to be closer to the desired output for each training example.

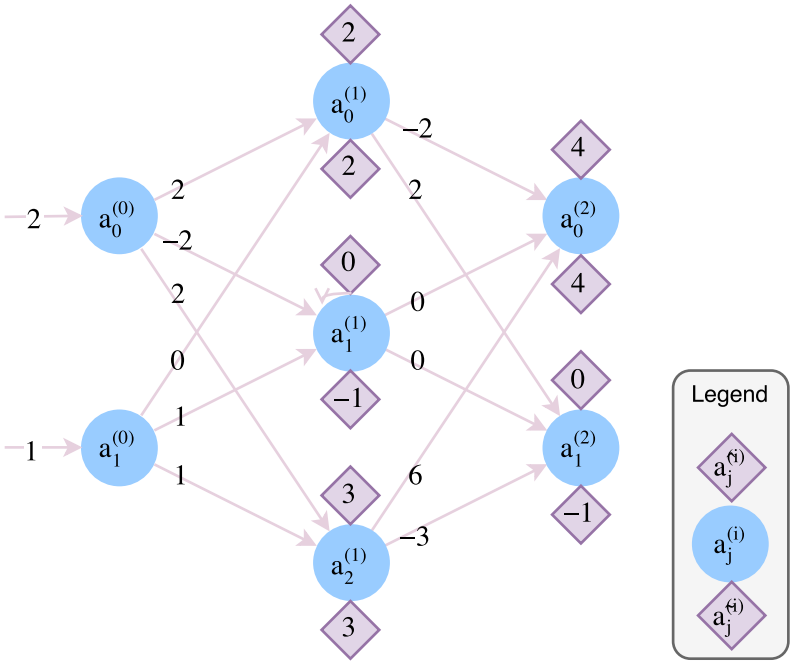


Figure 2.5: Forward propagation of the example ANN

The MLP is the foundation of modern neural networks, although in modern parlance it is known as a *fully connected feedforward network*. The network is *fully connected* since each neuron in layer $l+1$ is connected to every neuron in layer l. A *feedforward* network is one which neurons are connected in such a way that no cycles are present (networks with cycles are known as *recurrent networks*).

When producing any predictive model it is important to be able to evaluate it to determine whether it performs sufficiently well for its use case. There are many measures to evaluate models including (but not limited to): accuracy, precision, recall, and the F_1 score; picking a measure depends on the class ratio of the data you expect to run your model on and the cost ratio that defines how costly it is to mistake one class for another. Let's assume we've chosen accuracy to evaluate a perceptron we've just trained, to evaluate it we could see how it performs on the training data however, since we know that the perceptron perfectly splits the training data into two classes (otherwise the algorithm doesn't terminate) the training accuracy will always be 100%, making this a useless test. Instead, we need a new dataset (perhaps some data kept back from the training set) on which the model hasn't been trained, referred to as the *validation dataset*, on which we evaluate the performance.

2.2 Convolutional neural networks (CNNs)

CNNs, a specialised form of ANNs, were first proposed in [11] as a network architecture called the *neocognitron* inspired by the research of Hubel and Wiesel on the visual cortex[14]. Hubel and Wiesel found that the neurons in the visual cortex of a cat's brain responded to patterns in regions across the cat's field of view. They termed the region causing an excitation of a neuron as the *receptive field* of that neuron. Furthermore, they discovered that neurons were arranged in such a way that neurons with similar receptive fields were also physically co-located in the cortex. fig. 2.6 illustrates an example field of view shown

to a cat where two neurons with receptive fields r_1 and r_2 both were excited by the white dot, their position in the visual cortex is correlated with position of their receptive fields. Fukushima *et al.* designed the connectivity of the neurons in the neocognitron to model the connectivity of the neurons in the visual cortex such that each neuron was connected to neurons in the previous layer to form a receptive field. This architecture is very similar to those currently in use.

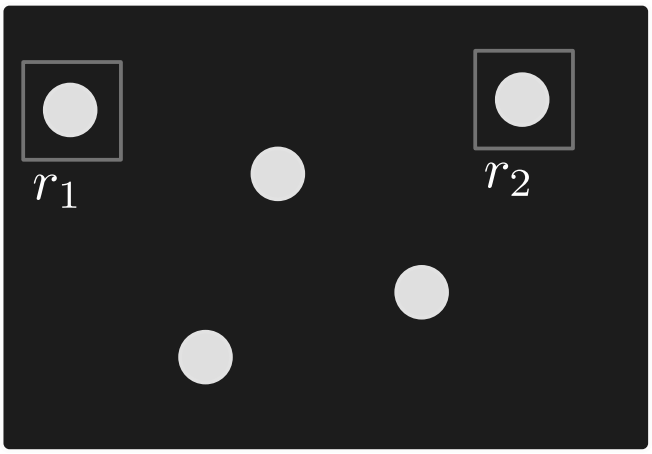


Figure 2.6: An image with white dots on shown to a cat in Hubel and Wiesel's experiments on the visual cortex. Grey rectangles r_1 and r_2 mark the receptive fields of two neurons that both detect the presence of a white dot in their respective receptive field

Building on the work of the neocognitron, modern CNN models introduce one substantial improvement: rather than learning parameters for each individual neuron in a layer, we instead assume that there exist neurons that fire in response to a class of patterns in the input for all receptive fields, this allows us to learn only a single set of weights for identifying the pattern in a receptive field, then those weights can be duplicated across all receptive fields producing a layer of neurons that respond to a given pattern at different locations in an input. Consider a layer of neurons that recognise the white dots in fig. 2.6 at different receptive fields, the weights of the connections going into a neuron in the layer is the same for all neurons throughout the layer, it is only the connectivity that differs (which area of the input image the neuron looks for the white circle). This layer architecture corresponds to convolution of a filter/kernel over the layer input, so we now switch to discussing CNNs from this viewpoint: learning filters and convolving them over the input to produce an output. The size required to store the convolutional layer parameters is massively reduced by only storing kernels instead of the duplicated kernels that correspond to each neuron for each receptive field.

The restricted architectures of CNNs facilitates a new view of these networks compared to ANNs; the overarching theme is to raise the level of abstraction from neurons to layers, and individual scalar inputs to tensors (N dimensional matrices). ANNs have no fixed/predetermined function, different groups of neurons in a layer can serve different purposes, however this is not the case in CNNs, layers are homogeneous in their function, e.g. a convolutional layer only computes the convolution of its input with a set of filters. CNN architectures are described by their constituent layers and the hyperparameters that configure those layers, different layer types have different hyperparameters.

Layers are constructed using this conceptual model and can be mapped down to the traditional ANN model of neurons and weights.

Inputs and outputs from layers are thought of as tensors rather than sets of disparate features encoded in a vector, this view is enabled the homogeneity of the input. Typically for image and video processing networks, the input is a 3D tensor, where width, W, and height, H, correspond to the width and height of the input image, and the depth, D, of the block corresponds to the number of channels in the image (e.g. 3 for RGB images, 1 for grayscale).

2.2.1 Layers

Layers can be thought of a functions over tensors. A tensor of dimensions $W \times H \times D$ used as input by a layer is transformed to an output tensor of dimensions $W' \times H' \times D'$ where the new dimensions are a function of the input tensor's dimensions and layer's parameters.

There are many types of layers, but they mainly fit into four broad categories: *fully connected*, *pooling*, *convolutional*, and *activation*.

Fully connected

Fully connected layers are like those in a MLP where each neuron is connected to every neuron in the previous layer, see fig. 2.7. These layers are very large in parameters so are usually used further in the network when the input tensor size is considerably reduced. In CNNs, fully connected layers draw together high level features from regions that are spatially distant from each other, consider the task of detecting a bike in an image, if you have neurons that fire on wheels, there will be neurons that activate when wheels are present in different locations in the image, the fully connected layer will be able to draw together the wheel-neuron activations that are spatially separate and help discriminate images of bikes from images with wheels that don't share the same spatial relationship that wheels on bikes do.

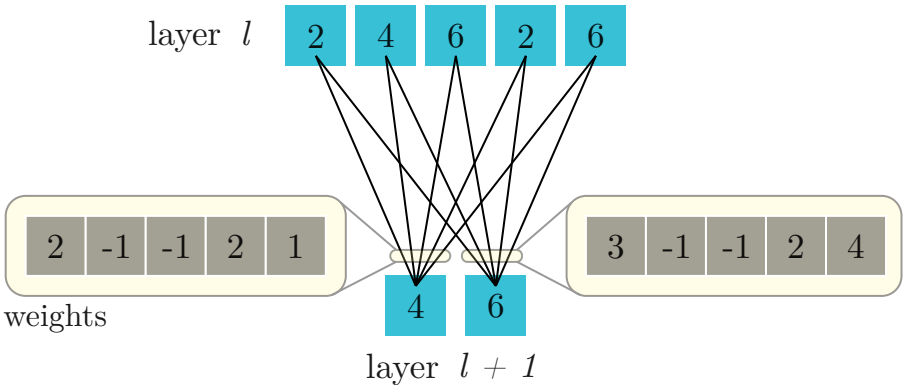


Figure 2.7: Fully Connected layer example. Weights of the edges are given by the corresponding component in the yellow boxes

Pooling

Pooling layers exist to increase the receptive field of deeper layers enabling them to learn features that span larger spatial extents, this is accomplished by reducing the size of the input tensor by computing some function over a region of the input yielding a single value. Max pooling is a common pooling filter where the maximum value in an input region is selected to be propagated forward discarding the rest of the values in the region, see fig. 2.8 for a small max pooling layer example.

Pooling layers typically have *size*, *pad* and *stride* parameters. The *size* determines the region over which pooling takes place, *padding* specifies the whether to zero pad the input along it's the borders of each axis and if so, how wide/deep the padding is, *stride* specifies how many elements to slide the filter along the input between each application of the pooling filter. For example, a 2D[3] max pooling layer with size 2×2, padding 1×1 and stride 2×2 will first pad it's input with a border of zeroes 1 element deep, then apply a 2×2 max pooling filter propagating the max value over that region to its output, it will shift by 2 elements right along the row and repeatedly be applied until reaching the end of the row then will be shifted 2 down, this process repeats until the filter reaches the bottom right most position. For an input tensor of size $224 \times 224 \times 10$ the padded input will be of size $226 \times 226 \times 10$, since the pooling region is 2×2 there are $(226 - 2) \times (226 - 2)$ locations it could be applied at, but our stride isn't 1×1, but 2×2 hence the output tensor will have dimensions $(226 - 2)/2 \times (226 - 2)/2 = 112 \times 112$

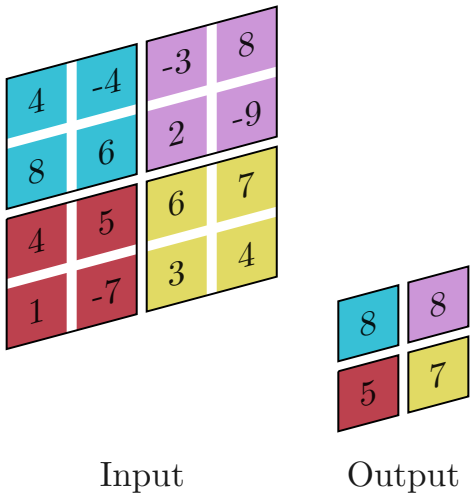


Figure 2.8: Max pooling layer example. Each coloured region in the input layer is processed by the max pooling function resulting in the component with the same colour in the output layer.

Convolutional

Convolutional layers consist of one or more filters that are slid along an input tensor and convolved at each location producing a single value which are aggregated in an output tensor, see fig. 2.9. The filter parameters are learnt and are constant across different locations in the input tensor, this massively reduces the number of parameters of the model

[3] 3D pooling is possible and used in some action recognition architectures

compared to a fully connected layer handling similar tensor sizes making them much more space efficient than fully connected layers.

The number of parameters in a convolutional layer is *only* dependent upon the filters. The filter is parameterised by its size, stride and zero padding. The *size* determines the dimensions of the tensor holding the filter weights; *stride*, how the filter is moved through the input tensor; *zero padding*, whether or not the input tensor is padded with zeros when convolved with the filter.

For a layer with N filters with the parameters:

- Size: $W_f \times H_f \times D_f = 3 \times 3 \times 1$
- Padding: $W_p \times H_p \times D_p = 0 \times 0 \times 0$
- Stride: $S_w \times S_h \times S_d = 1 \times 1 \times 0$

The layer has a total of $N \cdot W_f \cdot H_f \cdot D_f = 1 \cdot 3 \cdot 3 \cdot 1 = 3N$ parameters.

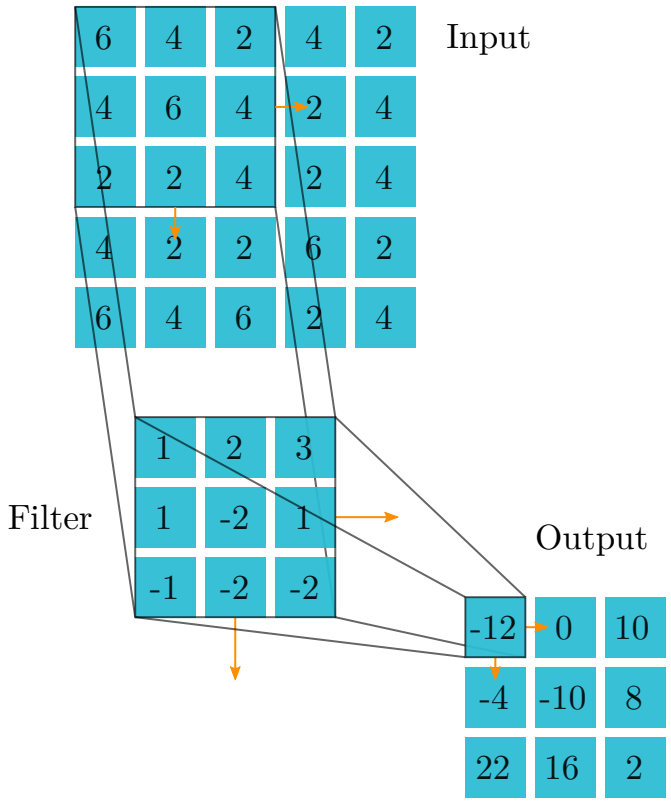


Figure 2.9: Convolutional layer example with a single filter. The filter has dimensions $3 \times 3 \times 1$, stride $= 1 \times 1$, and we do not use zero padding.

Activation

Activation layers are much the same as in traditional ANNs, an activation function is chosen and applied element wise to the input tensor to produce an output tensor of the same dimensions, see fig. 2.10. Activation functions take the form $\phi(x)$, common functions used include the rectified linear unit (ReLU), $\phi(x) = \max(x, 0)$; sigmoid, $\phi(x) = \frac{1}{1+e^{-x}}$; and hyperbolic tangent $\phi(x) = \frac{e^{2x}-1}{e^{2x}+1}$.

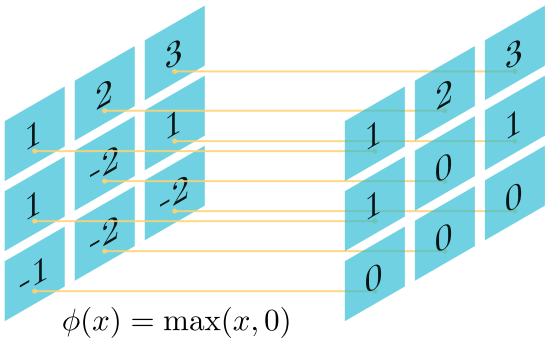


Figure 2.10: Activation layer example

2.2.2 Architectures

The architecture of a CNN refers to the choice of layers, parameters, and connectivity of those layers. Architectures are designed to solve a particular problem where the number of layers limits the complexity of the features learnt by the network. Networks are designed such that they have sufficient representational power (i.e. number of layers, and number of filters in those layers) necessary to learn the desired mapping from input to output, but are as small as possible as each new layer adds additional hyperparameters (number of filters, filter size, stride size) to the network which further increases the already considerable time spent searching for optimal hyperparameters by training a network per hyperparameter configuration.

First we look at architectures for object detection as this task has been the focus of most research, inspiring architectures for action recognition, which we discuss afterwards.

Object detection

Historically CNNs were extensively applied to the object detection problem popularised by the ImageNet challenge [35]. The challenge consists of two main problems: *object detection* and *object localisation*, participants produce a model capable of predicting the likelihood of the presence of a class of object (e.g. dog) in a test image for 1,000 different object classes. Models are evaluated based on their top-1 and top-5 error rate where the top-N error rate is defined as the proportion of test images whose prediction is considered an error if the ground truth label does not appear in the top-N predictions.

AlexNet

AlexNet[20] was the first CNN submission to ImageNet 2012 challenge achieving a large error-rate reduction over previous state of the art methods using feature engineering (unlike ANNs which learn feature representations from raw inputs) scoring a top-1 error rate of 36.7% and top-5 error rate of 15.3%.

In fig. 2.11 we present the architecture of AlexNet. The dimensionality of the tensors flowing through the network is gradually reduced by use of pooling layers. As the dimensionality of the inter-layer tensors decreases the convolutional layers learn features that span progressively larger spatial extents. Three fully connected layers are used to combine

features from locations across the input image. The final softmax layer acts as a linear classifier over the output feature vector from the last fully connected layer.

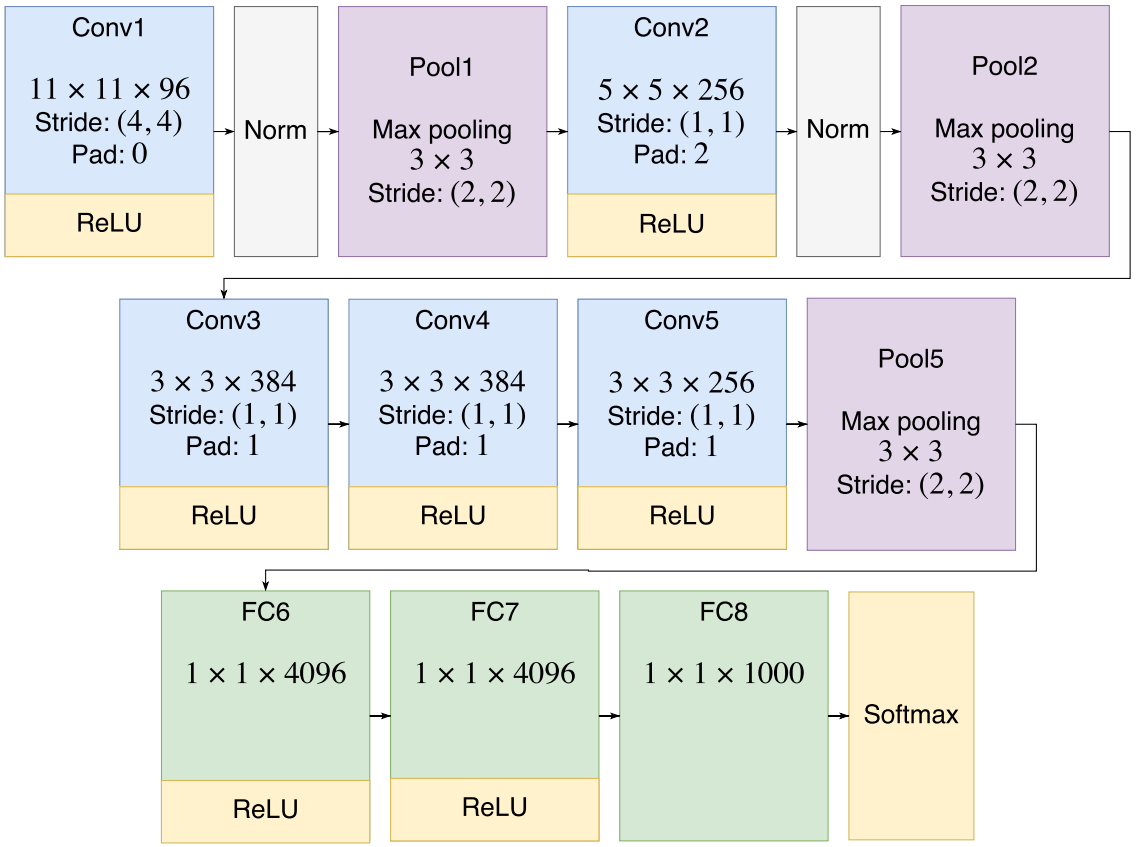


Figure 2.11: AlexNet Architecture

VGG16

The VGG16 architecture [41] was developed as an enhancement over the original AlexNet[20] architecture investigating the effects of using a 'very deep' architecture with many stacked convolutional layers. Six similar network architectures with increasing depth were trained and their classification performance tested against the ImageNet dataset. The network configurations with more convolutional layers performed better than those with fewer resulting in two configurations: VGG16 and VGG19 with 16 and 19 convolutional layers respectively. The VGG architectures (used in an ensemble) won first place in the object classification stream of the ImageNet 2014 challenge scoring a top-1 error rate of 24.4% and top-5 error rate of 7.0%, a considerable improvement over AlexNet. Another deep network architecture by Google[43] achieved similar error rates (second place) with 22 layers suggesting that more layers aren't necessarily better, but that the architectures in 2012 were too shallow.

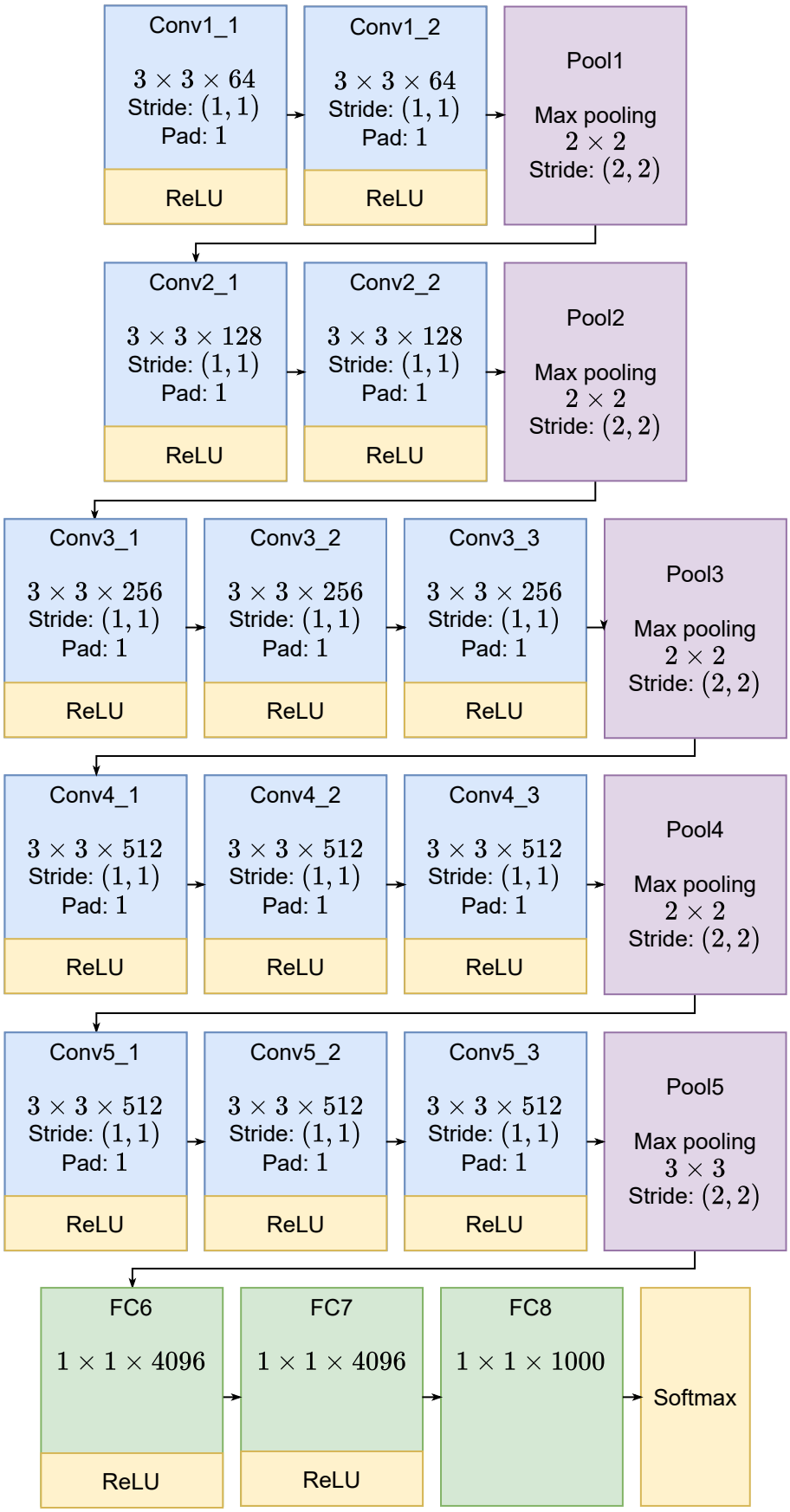


Figure 2.12: VGG16[41] Architecture

Action recognition

The challenge of recognising actions from video sequences has recently seen the application of CNNs inspired by their performance on object detection. A variety of architectures for tackling the problem have emerged which we shall explore in chronological order to see how architectures have evolved over time concluding with the architecture used in our experiments.

Figure 2.13: Which way is the tap turning? (BEOID)

Action recognition can be performed from single images or video sequences, both approaches have been investigated and we will outline prior work in this section. Some actions (or combinations of actions) cannot be reliable predicted by single images alone, this is because the motion in the action is the only distinguishing factor from other actions. For example, in fig. 2.13 a person turns a water tap in a kitchen sink; we cannot reliably determine whether the tap is being turned on or off from a single image alone. Instead it is necessary to examine multiple video frames over time to determine the direction of turning or whether water was flowing prior to turning the tap or not.

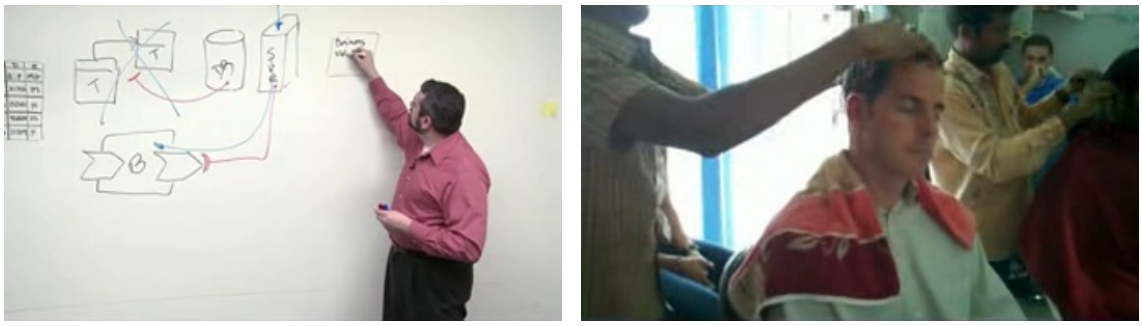


Figure 2.14: Contextual clues from the surrounding environment can aid action recognition from single frames

In contrast to the observation above, single images alone can perform well based on *cues* in the image. Consider the two images in fig. 2.14, the actions being performed can be determined from appearance alone, we can infer that the man is writing on the board because he is facing the board, the board has writing on it, and he is holding a pen close to where there is writing. Similarly, the head massage image is distinguishable from appearance alone providing there are not classes with similar appearances (such as *hair*

cut). Gkioxari *et al.* investigate the use of these contextual clues in designing an action recognition CNN architecture[12] obtaining state of the art performance on the Pascal VOC dataset[9].

The first investigation of CNNs for action recognition operating on raw frame data (i.e. without explicit feature extraction) was conducted by Baccouche *et al.* [1]. They introduced an architecture with an input of a stack of video frames which were then processed by multiple convolutional layers to learn spatio-temporal features on the KTH human actions dataset[38], the output of which was then used by a recurrent ANN architecture called long short-term memory (LSTM) to obtain a prediction over the whole video sequence, although they also compared classification with the LSTM layer to a linear classifier and only found modest performance benefits (on the order of a few percentage points) indicating that short clips from a video may be sufficient for action recognition in the KTH dataset.

A similar architecture with a larger input was investigated in [16], instead of training the whole network, the first layers were hand initialised to obtain the following transformations: rgb to grayscale, spatial gradients in both x and y directions, and optical flow in both x and y directions. A video sequence processed by the first layer results in a stack of grayscale video frames, spatial gradient frames and optical flow frames. The rest of the weights were trained by stochastic gradient descent. The network was evaluated on both the KTH dataset with competitive performance to other methods developed at the time, and TRECVID[44] dataset improving over the previous state of the art.

Two stream convolutional neural network (2SCNN)

A biologically inspired architecture based on the two-stream visual processing hypothesis for action recognition was introduced in [40]. The two stream hypothesis states that two processing streams are used in the brain for processing visual input: the *dorsal stream* for motion, good at detecting and recognising movements; and the *ventral stream* recognising form, good at detecting objects. The proposed model uses two separate CNNs each taking a different input based on the two stream hypothesis: the spatial stream for handling the appearance (analog of the ventral stream) and the temporal stream for handling the motion (analog of the dorsal stream). A video sequence is processed to obtain the optical flow frames which are used as input to the temporal stream, and a single frame is used as input to the spatial stream. The two streams process the inputs in parallel, and each stream produces an action prediction. The results are then combined using a linear classifier, see fig. 2.15 for a graphical depiction of the architecture and inputs to the streams.

The spatial stream takes in a single input frame of dimensions $W \times H \times 3$ (using RGB images) referenced by its index τ in the video sequence. The corresponding input to the temporal stream has dimensions $W \times H \times 2L$ where L is the temporal duration, a hyperparameter of the network determining the temporal window, the set of frames for which the network has information on. In [40] $L = 10$ is used (optical flow is thus calculated from 11 consecutive frames). The temporal input is computed from the frames $\tau - \lfloor (L+1)/2 \rfloor$ to $\tau + \lceil (L+1)/2 \rceil$ yielding $L+1$ frames from which $2L$ optical flow frames are obtained, twice as many as the input due to computing both flow in u and v directions. The flow frames are then combined by alternating u and v frames, all even frames in the optical flow stack are in u direction and odd, the v direction.

Raw optical flow frames stored as floats can take up a large amount of space so instead they are converted to greyscale images in the range $[0 \,..\, 254]$ and compressed using JPEG

to reduce storage requirements. On input to the network the frames are mean-centred around (127.5).

The networks are trained concurrently using mini-batch stochastic gradient descent. 256 video sequences are selected from the training dataset uniformly across the classes from which a single frame is sampled from each of these videos for which the corresponding optical flow stack is computed, forming the input of the spatial and temporal network respectively. Stochastic gradient descent with momentum is used to train the networks against the ground truth actions of the sample video. A common strategy in training CNNs is to initialise the weights of the network to those of the same network architecture trained on ImageNet helping to avoid overfitting on small datasets, Simonyan *et al.* apply this advice to training the 2SCNN.

Classification of a video using the two stream network is accomplished by sampling a fixed number of frames with equal temporal distance between each pair of consecutively sampled frames. For each sampled frame F, a new set of frames are computed by flipping and cropping F. The corresponding input to the temporal network is computed from the frames post-transformation. The overall class score for the whole video is computed as the average of the class scores for each individual sample. For example, sampling 20 frames from a 60 second long clip (at 24 FPS) will yield frames with indices $k \cdot 24 \cdot \frac{60}{20}$ for $k \in [0..19]$, each frame F_k at index k in the video will then be cropped and flipped to produce a set of derived frames $\mathcal{F}$, each frame in the set will be used to produce inputs to both the spatial and temporal streams. The scores for each frame in the $\mathcal{F}_{\mathcal{X}}$ are combined (*fused*) by a linear classifier to produce the final classification of the clip.

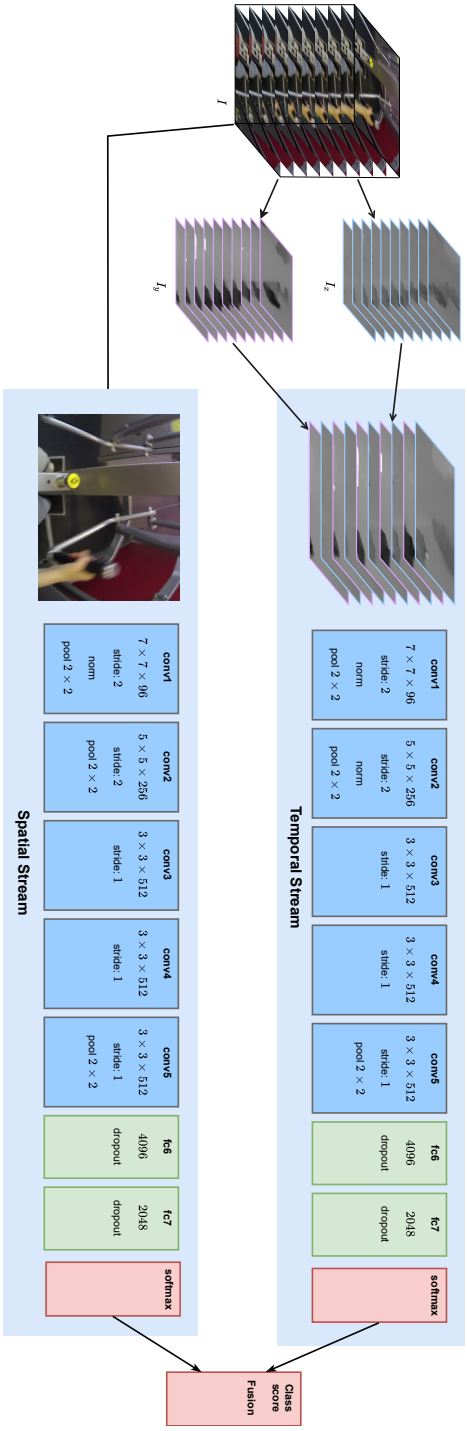


Figure 2.15: Two stream CNN architecture[40]

2.3 Video Datasets

In sec. 3 the surveyed papers frequently make reference to datasets, rather than explaining them as they are referenced they are instead described and consolidated in this section.

2.3.1 KTH - Human actions

The KTH human action[38] dataset is composed of 6 action classes: walking, jogging, running, boxing, hand waving and hand clapping performed by 25 subjects in 4 scenarios: outdoors, outdoors with scale variation, outdoors with different clothes, and indoors. Each action class has 100 example clips

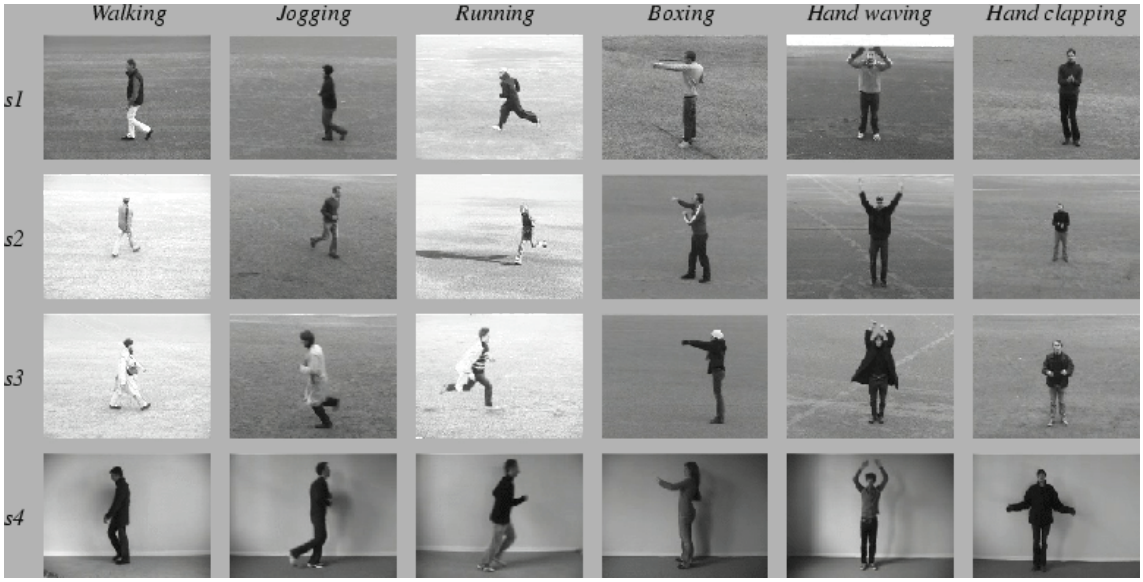


Figure 2.16: KTH Human actions[38] sample

2.3.2 TRECVID - London Gatwick Airport Surveillance video

TRECVID[44] is a competition held each year by The National Institute of Standards and Technology. In 2008 one of the challenges held asked participants to detect 10 different events in 100 hours of CCTV camera footage shot inside London Gatwick Airport[33]. The events to detect were: person puts mobile phone to ear; elevator doors opening with a person waiting in front of them, but the person doesn't get in before the doors close; someone drops or puts down an object; someone moves through a controlled access door opposite to the normal flow of traffic; one or more people walk up to one or more other people, stop, and some communication occurs; when one or more people separate themselves from a group of two or more people, who are either standing, sitting , or moving together communicating, and then leaves the frame; person running; person pointing; person taking a picture (descriptions taken from [33]).

2.3.3 Sports-1M - YouTube sport actions

Sports-1M[18] is a weakly annotated action dataset obtained from YouTube consisting of 1 million videos over 487 sport classes. The videos are obtained by searching for the sport class and then collecting videos from the search results hence the labels in the dataset are noisy[4].

2.3.4 HMDB51 - Human motion database

HMDB51[21] is a human activity dataset containing 6849 video clips over 51 action classes each containing a minimum of 101 clips each fitting one of 5 broad categories: general facial actions, facial actions with object manipulation, general body movements, body movements with object interaction, body movements for human interaction. Examples are given in fig. 2.17[5]

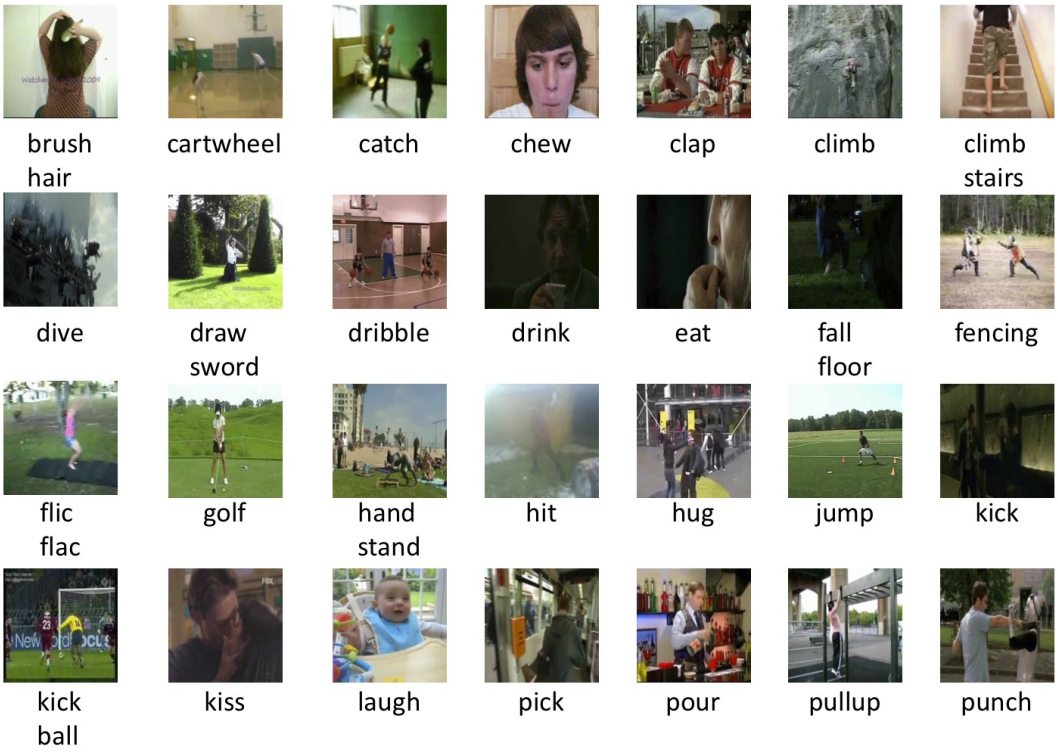


Figure 2.17: HMDB51[21] actions sample

2.3.5 UCF101 - Action recognition

UCF101[42] is an action dataset composed of 101 different actions across 5 broad categories: human-object interaction, body-motion only, human-human interaction, playing musical

[4]There exists incorrect labelled examples in the dataset
[5]HMDB51 Samples image obtained from http://serre-lab.clps.brown.edu/resource/hmdb-a-large-human-motion-database/

instruments, and sports. Each action class has a minimum of 100 example clips associated with it. The dataset has a diverse range of camera viewpoints, camera motion, object appearance and pose, illumination conditions making it quite challenging compared to some of the earlier datasets used for action recognition like KTH.

Figure 2.18: UCF101[42] actions sample

2.3.6 BEOID - Bristol Egocentric Object Interaction Dataset

BEOID[3] is an human-object interaction dataset composed of videos shot from a head mounted (egocentric) camera where the operator performs actions in one of 6 different locations: kitchen, workspace, printer, corridoor with locked door, cardiac gym, and weight-lifting machine.

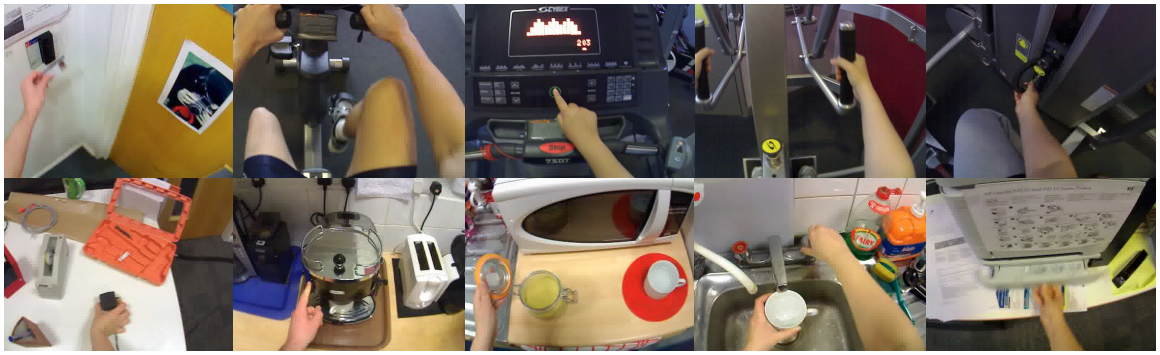


Figure 2.19: BEOID[3] object interactions sample

3 Understanding CNNs through visualisation

It is typical for CNNs to have on the order of 10^7–10^9 parameters, with this complexity comes a difficulty in understanding how the network works. There is a need to understand why a network correctly classifies some examples but not others to aid the researcher in determining higher performing architectures, and problems in the dataset or training process.

There is a substantial body of work CNN visualisation techniques and at first glance it can seem there are many different methods, however most methods fit into one of the following main categories of visualisation:

- **Filter analysis**, visualisation techniques based at the filter level: e.g. filter response and filter visualisation.
- **Attention mapping**, generating a heatmap over the input indicating which regions contribute to activation of a certain neuron or set of neurons.
- **Feature map inversion**, given a *feature map* (the output values computed at specific layer) what is an input to the network that produces in this feature map.
- **Activation optimisation**, given a neuron, or set of neurons, determine an input to the network that maximally or minimally excites the neurons.
- **Dataset-centric visualisation**, a graphical display of examples from the validation dataset aiming to provide a view of the examples through the lens of the network.

3.1 Filter analysis

Filter visualisation is the process of taking a filter and visualising it as an image. Typically this is most useful at the first layer in the network where the input to the filters are the raw input to the CNN (e.g. images), this allows us to draw direct comparisons with other filters used on similar inputs like edge detection filters for images thus giving us some insights to what the first layer is doing. [53] visualises the first layer filters of AlexNet[20] demonstrating that there are a number of 'dead' filters, uniform filters that don't compute any useful transform. Zeiler *et al.* empirically establish a new architecture which learns fewer dead filters in the first layer using first layer filter visualisation to check this.

Filter response involves visualising the response of a filter after application to a specific input, similar to *filter visualisation* this gives us insight as to what transformation the filters are computing: edge detection, contrast encoding etc. This tends to be most useful at lower layers in the network where the outputs from each layer are still recognisable as some transformation of the input image. Yosinski *et al.* introduce a tool called Deep Visualisation Toolbox[1] in [49] capable of showing filter responses for arbitrary networks

[1] https://github.com/yosinski/deep-visualization-toolbox

and input images in realtime (futhermore the tool can also compute the deconvolution and sensitivity attention maps). Yosinski *et al.* emphasise the importance of analysing all filters simultaneously as individual filters shouldn't be considered on their own but in the context of all the filters of the layer as this is the way the next layer uses them.

3.2 Activation optimisation

Activation optimisation covers a broad range of visualisation techniques used to optimise the activation of a neuron, usually we want to maximally activate a neuron to determine what features it recognises. Formally, to find an image that maximally activates a single neuron. Let $h_j^{(l)}(\boldsymbol{x})$ define the activation of neuron j in layer l when the network forward propagates the input $\boldsymbol{x}$. Now we want to find $\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}} h_j^{(l)}(\boldsymbol{x})$, this is not necessarily unique as many different inputs can cause maximal activation of a neuron. Whilst this may seem like a fairly simple optimisation problem the main challenge is to produce a 'natural' looking image, without any form of regularisation this optimisation produces extremely noisy unrecognisable images. Most of the prior work on this technique involves the proposal of different priors used to constrain the optimisation to produce interpretable images.

Erhan *et al.* were the first to introduce *activation maximisation* for generating an artificial image to maximise the activation of a chosen neuron by performing gradient ascent on an input image[8]. The authors only constrict the optimisation problem by ensuring that $||\boldsymbol{x}|| = \rho$, where ρ is used to bound the magnitude of the generated image. To evaluate the method, they apply the technique to a deep belief network[13] (DBN) and a stacked denoising autoencoder[46] (SDAE) trained on the MNIST dataset of 60,000 hand written digits. Images generated for neurons in lower levels showed blobs and simple patterns whereas neurons deeper in the network produced recognisable digit images indicating that the neurons have learnt higher level features from the combinations of lower ones.

Simonyan *et al.* investigate activation maximisation using L2 regularisation for object detection CNNs[39] producing images with interpretable outlines but inaccurate colouring.

Mahendran *et al.* investigate the use of priors to restrict the generated image to look 'natural' (i.e. not computer generated) for layer code inversion (see sec. 3.3). They use the L_n norm and *total variation* of the generated image[24] producing images with more accurate colouring than Simonyan *et al.*'s method. They later expand on their previous work in [25] recognising the general applicability of natural image priors to feature map inversion, activation maximisation and caricaturing. They note that pixels in a generated image should be constrained within a certain range, so propose a *bounded range* regularisation term to ensure the pixels are limited to be no greater than a chosen threshold. They also use *jittering*, as proposed in [15], which shifts the generated image between steps of the gradient ascent optimisation based on the assumption that the image should still produce a strong activation of the neuron if the edges are occluded. The effects of *jittering* and *total variation* regularisation can be seen in fig. 3.1.

Nguyen *et al.* propose an innovative method to encode the prior knowledge that the image should be 'natural' by use of a deep generative network (DGN) trained to invert a deep feature map back to image space[31]. A DGN G for a network N is trained to produce an input $\boldsymbol{x}$ to N from a feature map $\boldsymbol{m}$ from layer l in N such that forward propagating $\boldsymbol{x}$ in network N will produce $\boldsymbol{m}$ at layer l, i.e. the goal of a DGN is to invert a feature

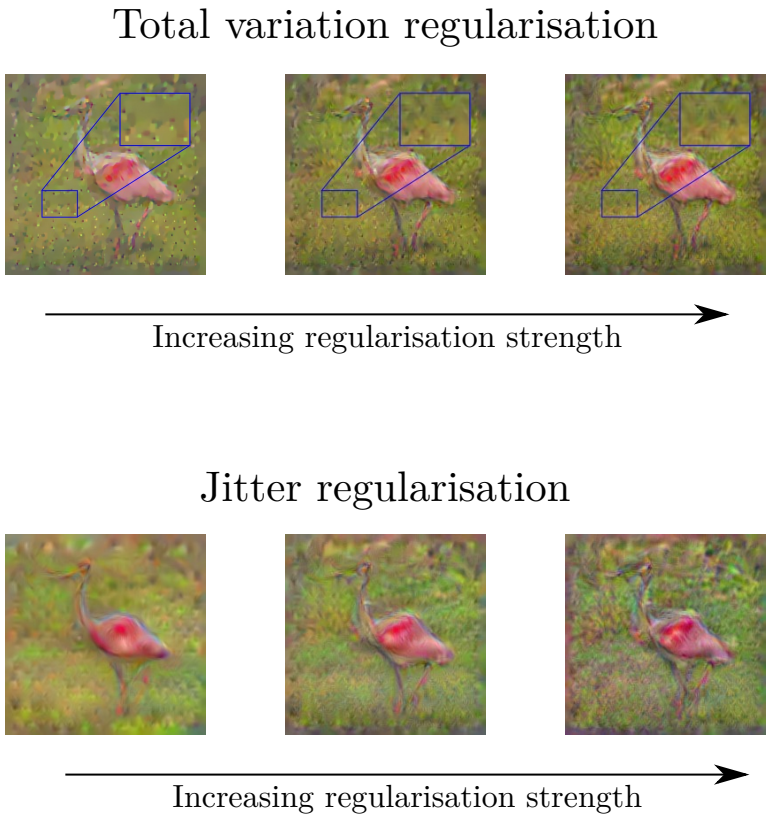


Figure 3.1: Demonstrating the effects of different regulariser in activation maximisation (images from [25])

map back to input space for a given network. Instead of performing gradient ascent in the input space to generate an input $\boldsymbol{x}^*$ to maximally activate $h_j^{(l)}(\boldsymbol{x})$ they instead perform it in the feature map space and connect the output of G to N forming a chain of the two networks. The use of the DGN acts as a strong prior for producing natural inputs since the DGN has in effect learnt what makes an image 'natural' or not.

Nguyen *et al.* also make an interesting discovery of multi-faceted neurons[30], neurons which are activated from multiple features and introduce a method to search for the different features that activate the neuron.

Whilst the previous work in activation optimisation has focused on improving the quality of generated images, Nguyen *et al.* have also investigated producing images of abstract patterns that cause unexpectedly high confidence classifications[29] yet to the human eye look nothing like the classes the network claim them to be.

Caricaturing (a.k.a Deep Dreaming) proposed by the Google AI team is very similar to *activation maximisation*, but rather than synthesising an artificial image by initialising the input as a random image, an image provided by the researcher. Activation maximisation is then carried w.r.t a chosen neuron resulting in a *caricature* of the original image in which regions contributing to the activation of the chosen neuron(s) are distorted to cause a higher activation. The distortions of the image tend to emphasize visual features relevant to the chosen neuron, but can also result in synthesising recognisable visual features out of nothing.

A visual comparison of results of the main methods for activation maximisation is presented in fig. 3.2

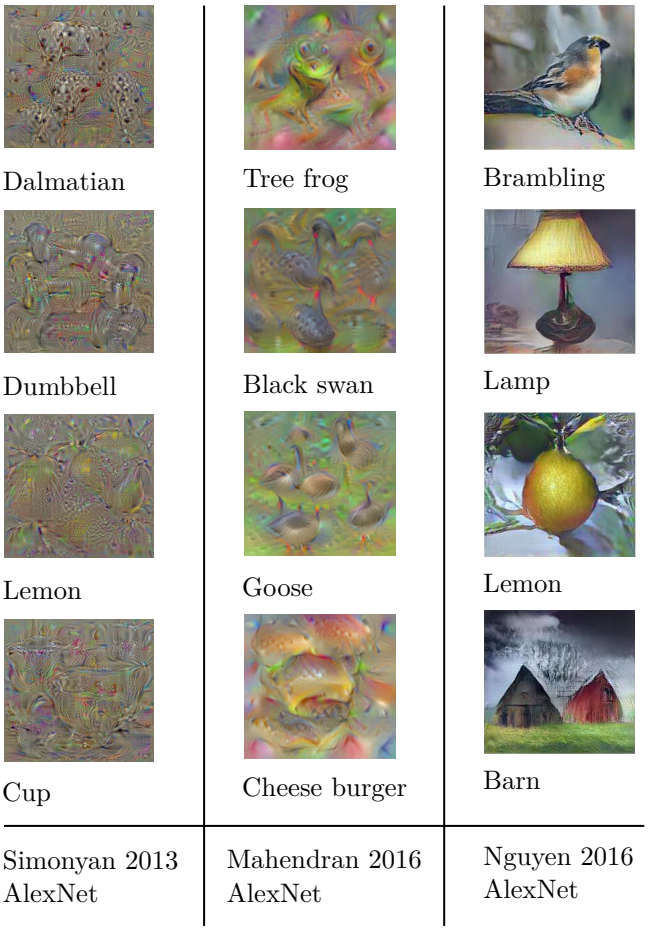


Figure 3.2: A comparison of the results of different methods for activation maximisation, the generated images come from the authors respective papers[39], [25], [31]

3.3 Feature map inversion

Feature maps (a.k.a activation map, CNN code, layer code) are the outputs produced by a layer for use by the next layer. In feature map inversion we try to determine an input $\boldsymbol{x}$ to the network to produce a given feature map $\boldsymbol{m}$ at layer l. This can be seen as a generalisation of activation maximisation where activation maximisation is feature map inversion for one-hot feature maps.

Mahendran *et al.* were the first to investigate the inversion of feature maps in object detection CNNs[24] using gradient descent to solve the following minimisation problem: For an input $\boldsymbol{x}$ which forward propagated through the network to layer l produces a feature map $\boldsymbol{m}$, find an input $\boldsymbol{x}^*$ from inverting $\boldsymbol{m}$ minimising the loss $\ell(\boldsymbol{x}, \boldsymbol{x}^*)$. Similar to activation maximisation, they also make use of priors to ensure the inversion looks like a 'natural' image. The priors to enforce 'natural' images previously discussed are just as applicable to feature map inversion as they are to activation optimisation.

Dosovitskiy *et al.* use *up-convolutional networks* to invert feature maps [6]. The authors train a decoder network on ImageNet images and feature maps from AlexNet[20] producing an up convolutional network. They contrast their results with Mahendran *et al.*'s method and decoders (as part of an autoencoder network where the encoder, AlexNet, is fixed) are trained for each layer in the network. Their results are presented in fig. 3.3

In [31] Nguyen *et al.* propose a DGN for use as a natural image prior in activation maximisation, however there is no reason that this couldn't also be used in the same manner as the up-convolutional decoder network for feature map inversion. However, this would be pointless in the case that *the feature map to invert comes from the same layer as the one that the DGN is trained to invert*, in that case the DGN would be used directly to invert the feature map.

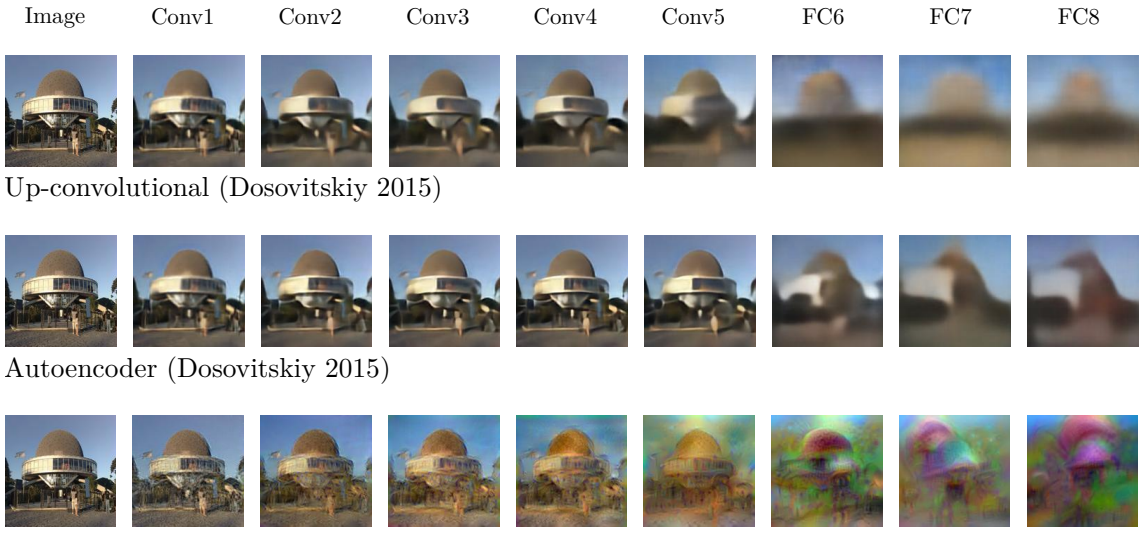


Figure 3.3: A comparison of different approaches for feature map inversion, the generated images all come from [6]

3.4 Attention mapping

An *attention map* for an input $\boldsymbol{x}$, and a neuron $a_j^{(l)}$ in a trained CNN is a heatmap indicating the regions in the input that contribute to the activation of $a_j^{(l)}$.

Occlusion study, Zeiler & Fergus[53] propose a simple method but computationally expensive method for determining the activation of a chosen neuron. For a specific input $\boldsymbol{x}$, a region is occluded by a mask smaller than the input size, a forward pass is computed, and the activation of the chosen neuron is measured. The occluding region is then slid along the input into a new position, and the process repeated over the whole input producing a tensor recording the neuron activation at each location which can then be used to produce a heat map. The computational cost of the method comes from forward propagation so frequently, once per occluding mask position. For a $m \times n$ heat map, the network is forward propagated $m \times n$ times.

Sensitivity analysis, Simonyan *et al.*[39] observe that the weights of a linear classifier can be interpreted as the relative importance of the components feature vector. Since CNNs compute a non-linear transformation of their input, the same technique cannot be used. A linear approximation $\hat{\boldsymbol{w}} \cdot \boldsymbol{x}$ to the network about a specific input $\boldsymbol{x}$ can be computed using a first order Taylor expansion whose weights $\hat{\boldsymbol{w}}$ can be interpreted as the importance of the corresponding input elements.

Deconvolution, In [53], Zeiler & Fergus introduce deconvolutional visualisation in which an input is propagated through the network, the neuron for visualisation is chosen and a deconvolutional network[52] constructed from the network-under-analysis' weights is attached to the layer in which the neuron of interest resides. All other neurons in the layer of the chosen neuron are set to zero to produce a one-hot feature map used as input to the deconvolutional network. The deconvolutional network progressively inverts the operation of the original network until the feature map has been propagated back into image space. The resulting image retains aspects of the original image in areas that contribute to the activation of the chosen neuron. To invert a convolutional layer l_c, a corresponding convolutional layer l'_c is constructed in the deconvolutional network where the filters from l are transposed in l'_c and the input to l'_c is the output of l_c. Rectified linear unit (ReLU) layers are inverted by also applying ReLU. The idea being that a ReLU layer ensures that the output of the layer is non negative, to preserve this property that the output of a layer is non negative in the deconvolutional network, we too have to add a ReLU layer. Pooling layers are inverted by recording the location in the filter from which the max activation originated from, consider the following example: in a pooling layer with 2×2 filters, index each location in the filter by i, let $i_{\max}$ by the index of the location from which the maximum value originates. When inverting the network, the value to by distributed back to the 2×2 grid is entirely given to location $i_{\max}$. Yu *et al.* make a qualitative comparison in [50] between AlexNet[20] and VGG16[41] using Deconvolutional visualisations of neurons in different layers showing that the deeper layers in VGG16 learn more discriminative features than those in AlexNet.

Layerwise relevance propagation (LRP)[2] produces discriminative attention maps highlighting regions which contribute to the classification of the input over other classes using a backprop like method. In [36], Samek *et al.* compare sensitivity analysis, deconvolution and LRP for object detection networks. One of their comparisons is reproduced in fig. 3.4

Excitation backprop[54] is addressed in depth in the following section (sec. 4).

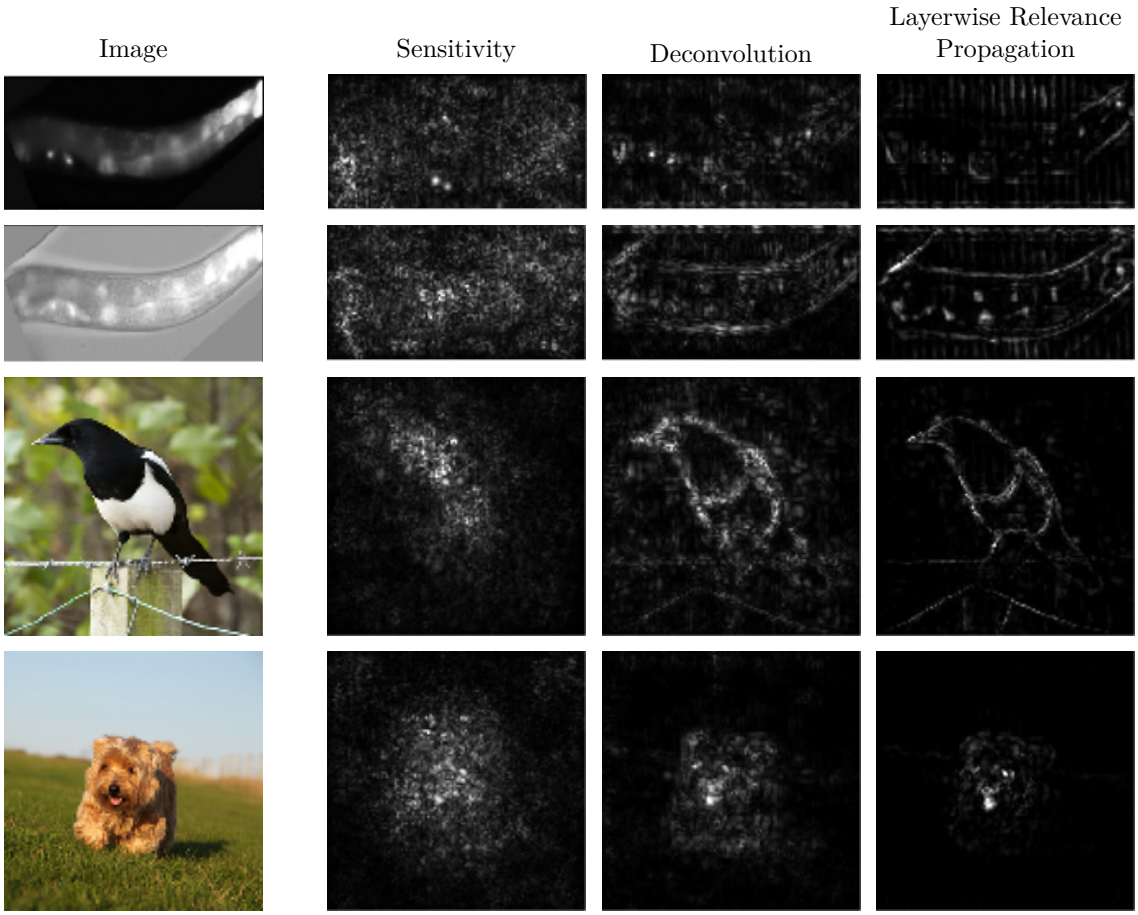


Figure 3.4: Samek *et al.*'s comparison of attention mapping methods from [36]

Evaluating attention maps The following methods for evaluating attention maps apply to object detection networks. [39] evaluate attention maps by using them to initialise an object segmentation algorithm from which they compute an object's bounding box which is evaluated against the ground truth bounding boxes indicating the network's ability to localise objects (a prerequisite for learning to recognise the object). [54] evaluate attention maps for object detection networks in 3 ways: *the pointing game* where a complex visual scene is used as input and the attention maps for each object (with corresponding class neuron) in the image is computed, the maximum from the attention map is extracted and checked to be inside or outside the bounding box of the object of interest. The number of hits and misses are counted (peaks inside and outside the bounding box); *object localisation* similar to [39], but instead of using an object segmentation to produce a bounding box they simply threshold the attention map and compute the smallest bounding box around the remaining thresholded points.

3.5 Dataset-centric visualisation

Dataset-centric visualisation methods all make use of examples from the validation set. The emphasis is on selecting or arranging examples to help determine network invariants.

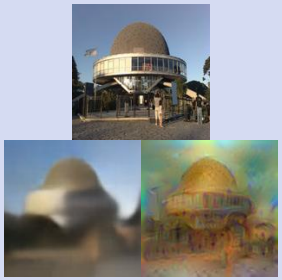
Dataset clustering through dimensionality reduction t-SNE (t-Distributed Stochastic Neighbour Embedding)[23] is an algorithm for dimensionality reduction where clusterings in the high dimensional space are preserved in the lower dimensional space. The method sees significant use in representing high dimensional data points in 2D. Andrej Karpathy uses t-SNE to visualise a subset of ImageNet validation images according to the corresponding FC7 feature maps from AlexNet, his visualisation is reproduced in fig. 3.5.

Example activation optimisation A simple technique for gauging what features a neuron might have learnt is to determine the top-n examples that minimally or maximally excite a chosen neuron. Through a comparative qualitative analysis of the examples one can determine the invariants of the neuron.

To conclude the section we give a one page overview of the different visualisation methods presented in fig. 3.6.

Figure 3.5: t-SNE visualisation of ImageNet validation images. Images close together have similar FC7 feature maps

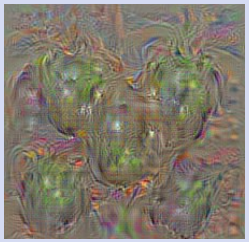
Feature map inversion



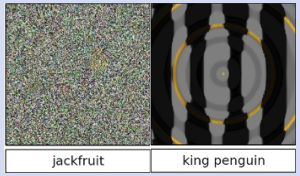
Inverting conv5 (AlexNet)
(Left - Dosovitskiy & Brox, 2015)
(Right - Mahendran & Vedaldi, 2014)

Activation optimisation

Activation optimisation using DGN as natural prior
(Nguyen et al, 2016)



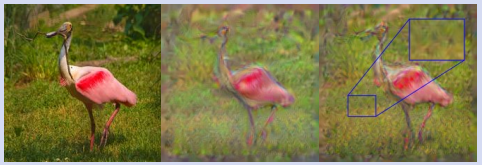
Activation maximisation
(Simonyan et al, 2013)



Fooling DNNs (Nguyen et al, 2014)

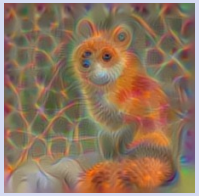
Activation maximisation
(Erhan et al, 2009)

Natural prior regularisation



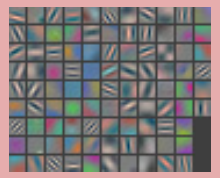
Inverting relu3 (AlexNet) feature maps, regularisation effects:
Left: Jitter, Right: Total variation
(Mahendran & Vedaldi, 2014)

Caricaturing

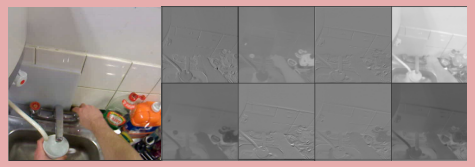


Caricaturing "red fox"
at layer relu4 (VGG-M)
(Mahendran & Vedaldi, 2016)

Filter Analysis

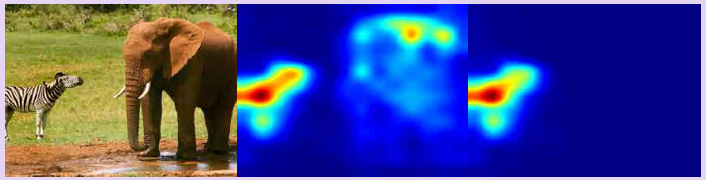


AlexNet Filter weights
(Zeiler & Fergus, 2013)

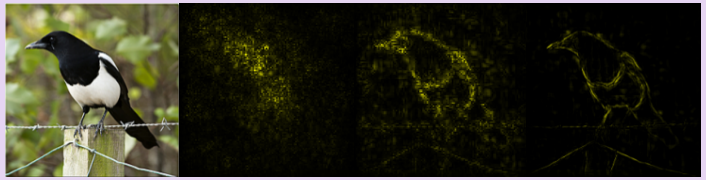


Filter responses of VGG-16 conv1 obtained with Deep Visualisation Toolbox
(Yosinski et al, 2014)

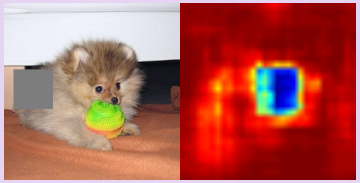
Attention mapping



Excitation backpropagation, non-contrastive (left), contrastive (right)
(Zhang et al. 2016)



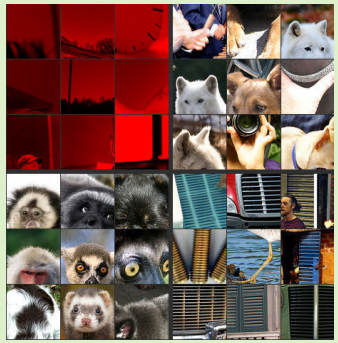
Saliency, Deconvolution, LRP attention maps
(Simonyan 2013, Zeiler & Fergus 2013, Bach et al. 2015)
(figure by Samek et al. 2015)



Occlusion study
(Zeiler & Fergus, 2013)

Dataset-centric

t-SNE ImageNet validation images
(visualisation by Andrej Karpathy)
(t-SNE by Maaten et al. 2008)



Example activation maximisation
top-9 examples for 4 neurons
(Zeiler & Fergus, 2013)

Figure 3.6: CNN visualisation technique summary

4 Excitation backpropagation

Excitation backpropagation[54] (EBP) is an attention mapping method inspired by visual attention in the primate visual cortex. *Visual attention* is the mechanism through which information from the visual field is selected. Attention can be split up into two distinct processes: *top down attention* and *bottom up attention*[4]. Bottom up attention is driven by raw sensory input shifting attention to potential regions of interests: regions that 'pop out' from the visual field, e.g. a shiny gold coin on the floor. Top down attention refines the input based on high level goals: regions satisfying the actor's search criteria e.g. spotting a friend in a crowd of people. Tsotos *et al.* propose a binary winner-takes-all (WTA) model of visual attention inspired by the primate visual cortex and the top-down bottom-up model of visual attention called the *selective tuning model*[45]. Zhang *et al.* adapt the selective tuning model of Tsotos *et al.* into a probabilistic formulation called Excitation Backpropagation capable of producing probabilistic attention maps instead of binary attention maps. The probabilistic attention maps confer the relative importance of regions in the input to a CNN based on a top-down attention signal provided by the researcher indicating their interest in activation of specific neurons in a chosen layer.

Like the top-down bottom-up model of primate attention, EBP decomposes attention into two parts: bottom-up and top-down. The model makes the following assumptions:

- The activation of neuron is positively correlated with the detection of a visual feature.
- The response of a neuron is non negative.

The bottom-up component of attention comprises the intermediate computations in the network modelling the intrinsic salience of the input. The top-down component, modelling the high-level search goal, is specified as a prior distribution over the neurons in the top layer. The prior distribution encodes the search goal as probabilities over the task relevant neurons, e.g. finding the discriminative regions in a video frame from a video of someone putting down a plug that cause the frame to be classified as 'put down plug' can be encoded as a one-hot probability distribution[1] over the classification layer of an action recognition network where all but the 'put down plug' class neuron probabilities are zero and the 'put down plug' class neuron is one.

Explanation EBP computes attention maps using a probabilistic winner-takes-all approach. A neuron is a winner neuron if it has the highest activation in the layer. *Winner-takes-all* refers to the winner neuron consuming all the attention from its children. The children of a neuron are the neurons in the previous layer connected to the winner neuron with non negative weights. At a high level, the idea is to consider each neuron in the top layer in turn, we assume that the neuron 'wins' and compute the conditional winning probabilities of each child neuron (neurons in the layer below connected to our winner neuron). The conditional winning probability describes the likelihood of a child neuron being a winner neuron in its layer conditioned on the knowledge that its parent is a winner

[1] We use the term one-hot probability distribution to refer to a distribution over a variable X in which $\exists x_{\text{hot}} \in X : P(X = x) = 1$ therefore $\forall x \in X \setminus \{x_{\text{hot}}\} : P(X = x) = 0$

neuron. Once we have computed the conditional winning probabilities of the children for each neuron in the layer, we then compute the marginal winning probability of each child neuron by marginalising the conditional winning probabilities of each neuron over its parents. We repeat the process by looking at the next pair of layers down; the previous bottom layer becomes the new top layer. This process is repeated until the marginal winning probabilities at the target stopping layer are obtained. See fig. 4.1 for a graphical explanation.

We now give a detailed explanation of EBP, interspersing the mathematical treatise with examples to aid exposition of the concept. The running examples are based on the following scenario: an image of a car is processed by an object detection CNN trained on ImageNet that correctly classifies the image as the class 'car'. We want to determine which regions of the input image contribute to its correct classification, i.e. what makes the image car-like?

For a given network, an input I is forward propagated thus computing the neuron activations $\hat{a}_j^{(i)}$ throughout the network. A layer L_{start} is chosen and a probability distribution $P(L_{\text{start}})$ encoding the relative interest in each neuron in the layer is defined. The probability distribution is defined:

$$P(L_{\text{start}}) = (P(a_0^{(l_{\text{start}})}), \ldots, P(a_n^{(l_{\text{start}})}))$$

L_{start} is a layer with index l_{start}[2], and $a_i^{(l_{\text{start}})}$ is neuron with index i in layer L_{start}. $P(a_i^{(l_{\text{start}})})$ is the marginal winning probability of the neuron $a_i^{(l_{\text{start}})}$, the probability that the neuron has the highest activation in the layer. In our example we define a probability distribution over the classification layer in which each neurons recognises a single object class in an image, the probability distribution is 'one-hot'; we set $P(a_{\text{car}}) = 1$ as we have no interest in any other object class.

The next steps are repeated for each layer in the network starting at the *start* layer and proceeding until the *stopping* layer L_{stop} is reached. At each step we consider two adjacent layers, the top layer (closer to the output of the network) L_{top} and the layer below L_{bottom} (closer to the input of the network). The probability distribution $P(L_{\text{top}})$ will always be defined during each step, and we will compute $P(L_{\text{bottom}})$ using the rules of EBP. For each neuron $a_j^{(l)} \in L_{\text{top}}$ we compute the marginal winning probability of its children $\mathcal{C}_j^{(l)}$ where

$$\mathcal{C}_j^{(l)} = \{a_k^{(l-1)} | w_{k,j}^{(l-1)} \neq 0\} \tag{4.1}$$

The conditional winning probability of a neuron $a_k^{(l-1)}$ given $a_j^{(l)}$ is a winning neuron is computed by

$$P\left(a_k^{(l-1)} | a_j^{(l)}\right) = \begin{cases} Z_j^{(l)} \hat{a}_k^{(l-1)} w_{k,j}^{(l-1)} & w_{k,j}^{(l-1)} \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

This formulation integrates the bottom up attention in the form of $\hat{a}_k^{(l-1)}$ which we assumed to be positively correlated with features present in the input I. The sum of the CWP for

[2]Layers are labelled bottom up, from input layer to output layer, starting at 0 (in contrast to Zhang *et al.*'s explanation[54]).

the children of $a_j^{(l)}$ isn't necessarily going to sum to one, so to make it a valid probability distribution we normalise by a factor $Z_j^{(l)}$ to ensure that $\sum_{a_k^{(l-1)} \in \mathcal{C}_j^{(l)}} P\left(a_k^{(l-1)} | a_j^{(l)}\right) = 0$

$$Z_j^{(l)} = 1 / \sum_{k: w_{k,j}^{(l-1)} \geq 0} \hat{a}_k^{(l-1)} w_{k,j}^{(l-1)} \tag{4.3}$$

having computed the CWPs for all parent-child pairs in L_{top} and L_{bottom} we then compute the MWP for each neuron $a_k^{(l-1)} \in L_{\text{bottom}}$ by marginalising (4.5) over the neuron's parents $\mathcal{P}_k^{(l-1)}$.

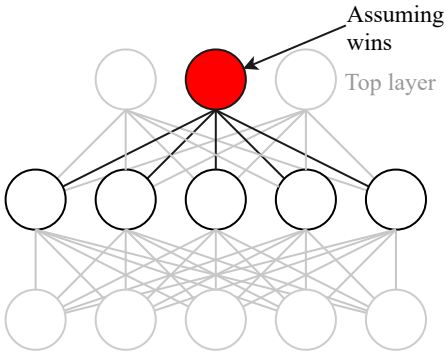
$$\mathcal{P}_k^{(l-1)} = \{a_j^{(l)} | w_{k,j}^{(l-1)} \neq 0\} \tag{4.4}$$

$$P\left(a_k^{(i)}\right) = \sum_{a_j^{(i+1)} \in \mathcal{P}_k^{(i)}} P\left(a_k^{(i)} | a_j^{(i+1)}\right) P\left(a_j^{(i+1)}\right) \tag{4.5}$$

The process is then repeated by treating L_{bottom} as the new L_{top} since we have computed the marginal winning probabilities for the layer and hence can compute the conditional winning probabilities for the next layer down. Once the target layer is reached the process is completed.

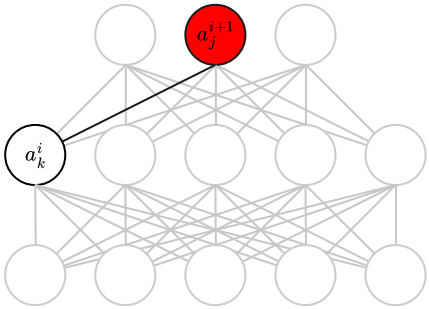
In summary:

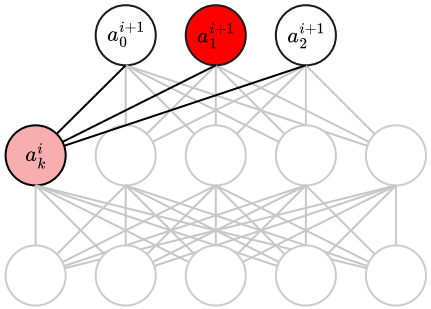
- Compute a forward pass of the network to determine the outputs of each neuron $\hat{a}_j^{(l)}$
- Iterating over pairs of layers top down from L_{start} until the lower layer becomes the target stopping layer L_{stop}
  - Compute the scaling factors $Z_j^{(l)}$ of each neuron in the upper layer.
  - Compute the conditional winning probabilities $P\left(a_k^{(l-1)} | a_j^{(l)}\right)$ of each neuron in the lower layer.
  - Compute the marginal winning probabilities $P\left(a_j^{(l-1)}\right)$ of each neuron in the lower layer by marginalising over its parents.



What are the probabilities of the neurons in this layer being winners, given we know the parent neuron is a winner?

The probability of a_k^i being a winner conditioned on a_j^{i+1} being a winner is dependent upon its activation $\hat{a}_k^i$, the weight linking it to its winner parent w_{kj}^i and the normalisation factor Z_j^{i+1} (computed from the weighted activations of all the children of a_j^{i+1}) to ensure $\sum_k P(a_k^i|a_j^{i+1}) = 1$





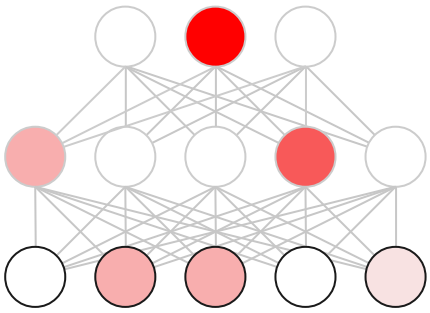
The marginal winning probability of a_k^i, $P(a_k^i|a_j^{i+1})$, is computed by marginalising the conditional winning probabilities of a_k^i over its parents $\mathcal{P}_k^i$:

$$\mathcal{P}_k^i = \{a_j^{i+1} | w_{kj}^i \neq 0\}$$

giving

$$P(a_k^i) = \sum_{a_j^{i+1} \in \mathcal{P}_k^i} P(a_k^i|a_j^{i+1}) P(a_j^{i+1})$$

Once the marginal winning probabilities for a layer are calculated, the process is repeated at the next layer down, until the marginal winning probabilities for the target bottom layer have been computed.



Colours indicate the magnitude of marginal winning probability

Figure 4.1: EBP in a nutshell

4.1 Contrastive EBP

In networks trained for classification, each class is represented by a different neuron in the final layer of the network, to determine the regions in the input that contribute to the activation of the class neuron we can model the top-down attention as a one-hot probability distribution. We can determine the regions of interest by using EBP and this prior distribution encoding the top-down attention yielding an attention map. This method has one main caveat: regions that increase the activation of one neuron may well increase the activation of other neurons (regions in which there are features common to the two classes). Depending on the goal of the user, this may be desired, or distracting, the user might wish to understand "why is this image of a cat classified as a cat and not something else?" in this case we want to encode the question "why is this classified as *cat* and not *non-cat*". This is a question of finding the discriminative features in the input, *contrastive* EBP proposed by Zhang *et al.* in [54] extends EBP to help answer this question. We have already created a distribution modelling *cat*, but to model a *non-cat* distribution we have to modify the network; we construct a new network where all the weights are the same apart from those in the last layer in which we invert all the weights to the class neurons, transforming positively discriminative neurons into negatively discriminative neurons (i.e. the *cat* neuron becomes *non-cat* in the new network). We can then compute the attention maps from both networks forming two attention maps: A_{pos}, an attention map that indicates the regions contributing to the *cat* classification and another A_{neg}, indicating the regions contributing to the *non-cat* classification. By subtracting the *non-cat* attention map from the *cat* attention map we end up with an attention map $A_{\text{contrastive}}$ describing the features in the input that contribute to the *cat* classification but not to anything else.

4.2 Worked example of EBP

First a forward pass of the network is computed, this produces the intermediate neuron values which are used as *bottom up* salience factors, then a probability distribution over the output layer is used to specify *top down* salience, then a excitation backprop pass uses the probability distribution, intermediate neuron values and weights to determine the probability of each intermediate neuron being a winner at an arbitrary depth of the network.

Contrastive top down attention uses the insight that we're not only interested in class we're localising, but also the absences of the other classes (as classes may be correlated), we EBP the class of interest one layer, then invert the output probability distribution, EBP one layer and compute the difference between the two MWPs of the second last layer, then EBP from there to the input.

We demonstrate EBP with a simple network composed of 5 neurons over 3 layers all using ReLU activations.

For a given neuron $a_j^{(l)}$, we denote the input to the neuron as $\tilde{a}_j^{(l)}$ defined in (4.6) and the output of the neuron as $\hat{a}_j^{(l)}$ defined in (4.7).

$$\tilde{a}_j^{(i+1)} = \sum_{a_k^{(i)} \in \mathcal{C}_j^{(i+1)}} w_{k,j}^{(i)} \hat{a}_k^{(i)} \tag{4.6}$$

$$\hat{a}_j^{(i)} = \phi(\tilde{a}_j^{(i)}) \tag{4.7}$$

Where ϕ is an activation, if not explicitly stated it is assumed $\phi(x) = \max(0, x)$ (ReLU activation).

Performing excitation backprop on the example network in fig. 2.4. The forward pass is detailed in fig. 2.5.

First we define the input of the network (these could be any arbitrary input):

$$\hat{a}_0^{(0)} = 2$$
$$\hat{a}_1^{(0)} = 1$$

Now we compute the forward pass using the forward propagation rule

$$\hat{a}_0^{(1)} = \max(0, 00 \cdot w_{0,0}^{(0)} + \hat{a}_1^{(0)} \cdot w_{1,0}^{(0)}) = max(0, (2 \cdot 1) + (1 \cdot 0)) = 2$$
$$\hat{a}_1^{(1)} = \max(0, (2 \cdot -1) + (1 \cdot 1)) = \max(0, -1) = 0$$
$$\hat{a}_2^{(1)} = \max(0, (2 \cdot 1) + (1 \cdot 1)) = 3$$

$$\hat{a}_0^{(2)} = \max(0, 10 \cdot w_{0,0}^{(1)} + \hat{a}_1^{(1)} \cdot w_{1,0}^{(1)} + \hat{a}_2^{(1)} \cdot w_{2,0}^{(1)}) = 4$$
$$\hat{a}_1^{(2)} = \max(0, (2 \cdot 1) + (0 \cdot 2) + (3 \cdot -1)) = 0$$

The next step is to compute the conditional winning probabilities of each neuron given each parent neuron wins using eq. 4.2, to compute this we need the scaling factors $Z_j^{(i)}$ which we will compute first using eq. 4.3 (in a computational implementation these would be computed on a per layer basis and discard once the layer values are calculated).

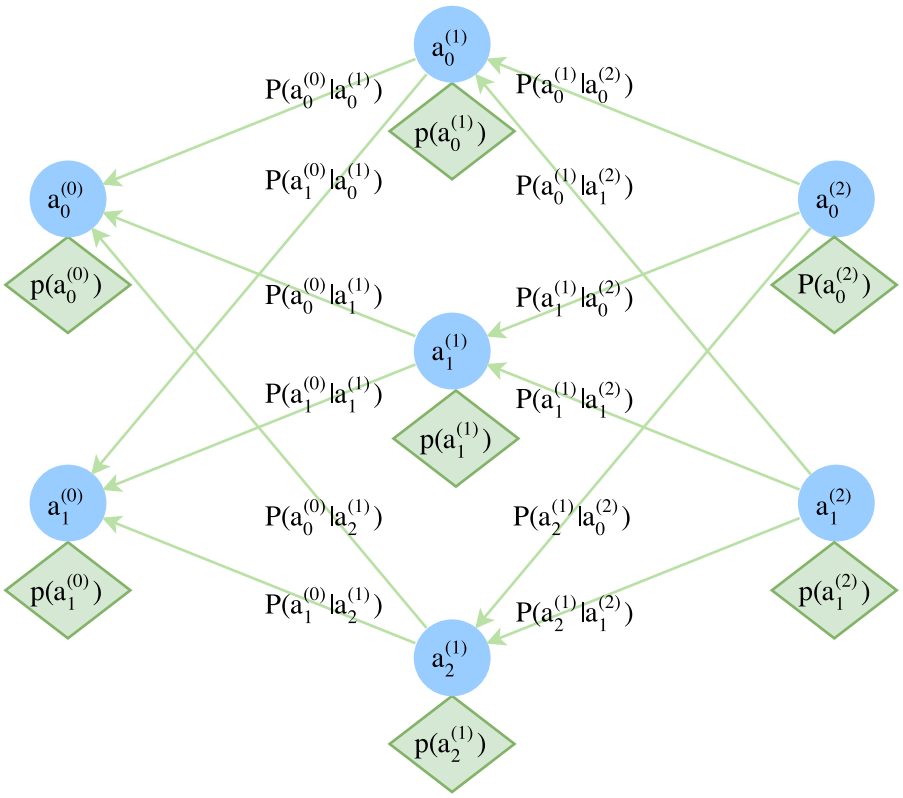


Figure 4.2: Flow of probabilities in EBP

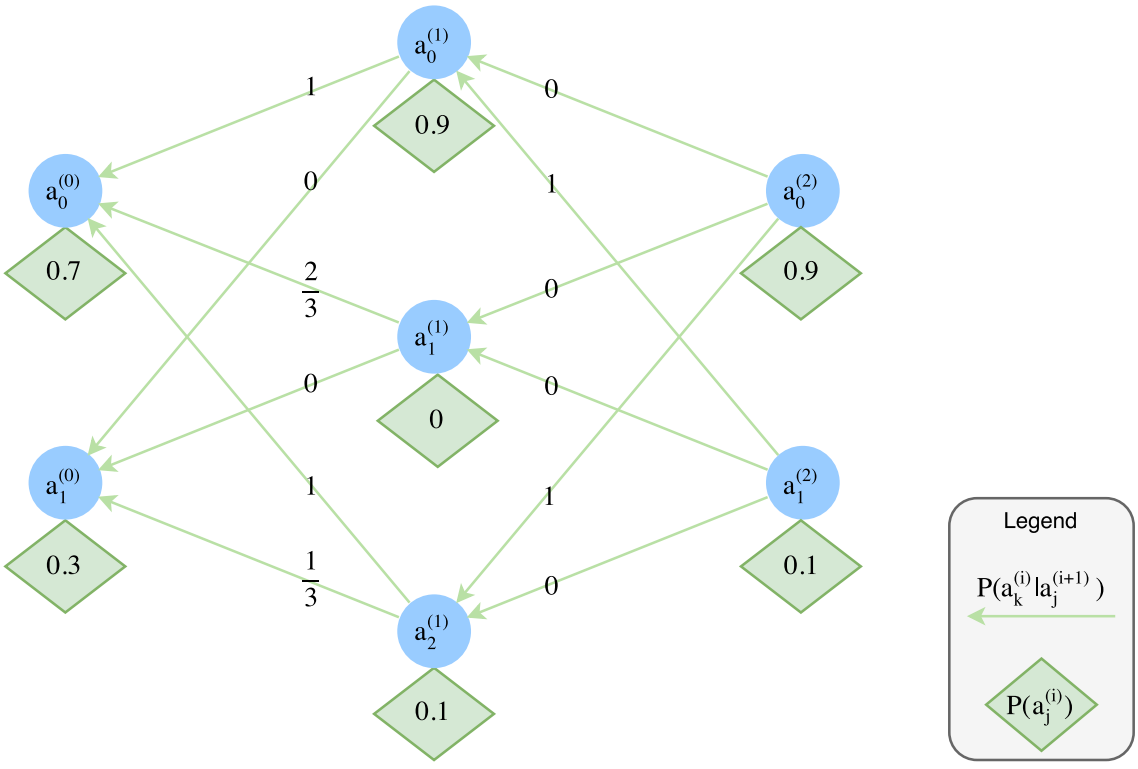


Figure 4.3: EBP CWP on the example network

$$
\begin{aligned}
Z_0^{(2)} &= \frac{1}{\left(w_{1,0}^{(1)}\hat{a}_1^{(1)}\right) + \left(w_{2,0}^{(1)}\hat{a}_2^{(1)}\right)} = \frac{1}{(1\cdot 0)+(2\cdot 3)} = \frac{1}{6} \\
Z_1^{(2)} &= \frac{1}{(1\cdot 2)+(2\cdot 0)} = \frac{1}{2} \\
Z_0^{(1)} &= \frac{1}{\left(w_{0,0}^{(0)}\hat{a}_0^{(0)}\right) + \left(w_{1,0}^{(0)}\hat{a}_0^{(0)}\right)} = \frac{1}{(1\cdot 2)+(0\cdot 1)} = \frac{1}{2} \\
Z_1^{(1)} &= \frac{1}{(1\cdot 1)} = 1 \\
Z_2^{(1)} &= \frac{1}{(1\cdot 2)+(1\cdot 1)} = \frac{1}{3}
\end{aligned}
$$

Now for the conditional winning probabilities between layers 2 and 1:

$$
\begin{aligned}
P\left(a_0^{(1)}|a_0^{(2)}\right) &= 0 \\
P\left(a_0^{(1)}|a_1^{(2)}\right) &= Z_1^{(2)}\hat{a}_0^{(1)}w_{0,1}^{(1)} = \frac{1}{2}\cdot 2\cdot 1 = 1 \\
P\left(a_1^{(1)}|a_0^{(2)}\right) &= Z_0^{(2)}\hat{a}_1^{(1)}w_{1,0}^{(1)} = \frac{1}{6}\cdot 0\cdot 1 = 0 \\
P\left(a_1^{(1)}|a_1^{(2)}\right) &= Z_1^{(2)}\hat{a}_1^{(1)}w_{1,1}^{(1)} = \frac{1}{2}\cdot 0\cdot 2 = 0 \\
P\left(a_2^{(1)}|a_0^{(2)}\right) &= Z_0^{(2)}\hat{a}_2^{(1)}w_{2,0}^{(1)} = \frac{1}{6}\cdot 3\cdot 2 = 1 \\
P\left(a_2^{(1)}|a_1^{(2)}\right) &= 0
\end{aligned}
$$

Now layers 1 and 0:

$$
\begin{aligned}
P\left(a_0^{(0)}|a_0^{(1)}\right) &= Z_0^{(1)}\hat{a}_0^{(0)}w_{0,0}^{(0)} = \frac{1}{2}\cdot 2\cdot 1 = 1 \\
P\left(a_0^{(0)}|a_1^{(1)}\right) &= 0 \\
P\left(a_0^{(0)}|a_2^{(1)}\right) &= \frac{1}{3}\cdot 2\cdot 1 = \frac{2}{3} \\
P\left(a_1^{(0)}|a_0^{(1)}\right) &= 0 \\
P\left(a_1^{(0)}|a_1^{(1)}\right) &= 1\cdot 1\cdot 1 = 1 \\
P\left(a_1^{(0)}|a_2^{(1)}\right) &= \frac{1}{3}\cdot 1\cdot 1 = \frac{1}{3}
\end{aligned}
$$

We can now marginalise over the parent neurons in the conditional winning probabilities if a prior distribution over the output neurons is given to obtain the marginal winning probabilities of each neuron using eq. 4.5.

Let's choose $P\left(a_0^{(2)}\right) = 0.9$ and $P\left(a_1^{(2)}\right) = 0.1$ for the prior distribution. If we were investigating the saliency of a single neuron we'd instead set the MWP of that neuron to 1 and the MWP of all other neurons would be 0.

Marginalising over the parents of the hidden layer:

$$\begin{aligned}
P\left(a_0^{(1)}\right) &= \sum_{a_j^{(2)} \in \mathcal{P}_0^{(1)}} P\left(a_0^{(1)} | a_j^{(2)}\right) P\left(a_j^{(2)}\right) \\
&= P\left(a_0^{(1)} | a_0^{(2)}\right) P\left(a_0^{(2)}\right) + P\left(a_0^{(1)} | a_1^{(2)}\right) P\left(a_1^{(2)}\right) \\
&= 0 \cdot 0.9 + 1 \cdot 0.1 = 0.1 \\
P\left(a_1^{(1)}\right) &= 0 \cdot 0.9 + 0 \cdot 0.1 = 0 \\
P\left(a_2^{(1)}\right) &= 1 \cdot 0.9 + 0 \cdot 0.1 = 0.9
\end{aligned}$$

Finally to calculate the MWP of the input neurons to obtain the posterior distribution:

$$\begin{aligned}
P\left(a_0^{(0)}\right) &= 1 \cdot 0.1 + 0 \cdot 0 + \frac{2}{3} \cdot 0.9 = 0.7 \\
P\left(a_1^{(0)}\right) &= 0 \cdot 0.1 + 1 \cdot 0 + \frac{1}{3} \cdot 0.9 = 0.3
\end{aligned}$$

5 Investigating features learnt by 2SCNN networks

We investigate the features learnt by two 2SCNN networks sharing the same architecture on two datasets: BEOID and UCF101. We produce and analyse the filters of the first and second layers of the networks. An extension of EBP is presented for use on the temporal stream of a 2SCNN to produce attention maps on a per frame basis. We produce attention maps using EBP from both the spatial and temporal streams and qualitatively analyse the output to determine features learnt by the network and pathological behaviour. The attention maps are quantitatively evaluated using two methods: for both network we evaluate the *jitter* across sequences of attention maps, and for the BEOID trained network we compare the maximum of the attention map to the action location using the operator's gaze as a proxy variable.

A good attention map will have maxima in the regions of the input that the network uses to discriminate between classes, helping to determine why the network has classified the input as it has, giving us insight into the salient features of the input with respect to a specific neuron (usually a class neuron). For example, consider an object detection network with a 'person' neuron that detects people in images, if we produce an attention map for the 'person' class neuron overlaid atop the image of a person used as input to the network, and the attention map highlights the person but not the surrounding background then we can conclude that the network has learnt how to recognise a person, at least in relation to the other classes the it is trained on. In contrast, if the attention map has no meaningful correlation with the regions in which the person is present, then there is little information we can derive from the attention map other than the network can't distinguish the person from its surrounding context.

For a network to have truly learnt to recognise actions without overfitting to a specific dataset it is necessary that the network should be able to localise the action; it is not possible to recognise an action without knowing where it has taken place. If the network can localise actions then the attention maps should be maximal in regions over the action. The following evaluations aim to quantify this property.

We produce videos with the overlaid attention maps for both the spatial and temporal streams, results are available on YouTube:

- UCF101 EBP videos: https://goo.gl/QBYZLJ
- BEOID EBP videos: https://goo.gl/PazivH

5.1 Networks

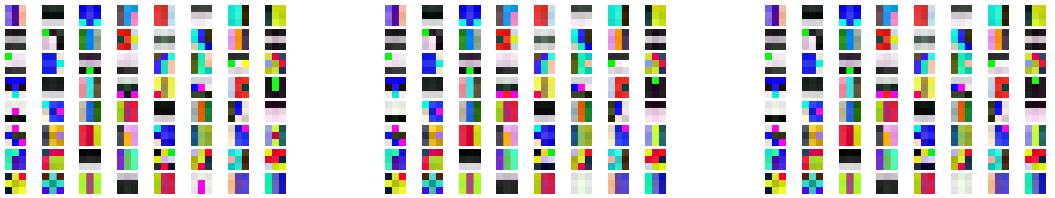
We investigate the application of EBP to a 2SCNN network constructed from two VGG-16 network towers forming the spatial and temporal stream, networks were provided pretrained

for use by CUHK[1] and UoB[2]. We use EBP to produce both non-contrastive and contrastive attention maps to help determine what features the networks learn to recognise.

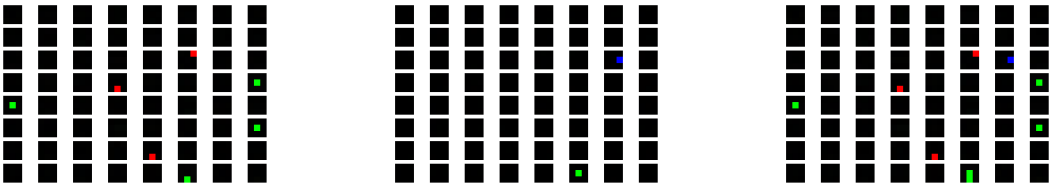
5.2 Learnt filter analysis

We present the learnt filters for the spatial stream of the 2SCNN for both BEOID and UCF101 in fig. 5.1, the filter sets are similar with only 1/64 of the parameters differing across the two spatial stream models. The filters for the action recognition networks are initialised to values from the same VGG16 architecture trained for object detection on ImageNet during training. We also present the filters for the VGG16 model trained on ImageNet for object detection in fig. 5.1c.

The filters for both the BEOID and UCF101 models are similar to the filters of the ImageNet model; we suggest that this is due to the spatial network predicting actions based on object recognition and so we see little divergence from the ImageNet model's filters.



(a) UCF101 `conv1_1` filters (b) BEOID `conv1_1` filters (c) ImageNet `conv1_1` filters



(d) Difference between UCF101 and ImageNet filters (e) Difference between BEOID and ImageNet filters (f) Difference between UCF101 and BEOID filters

Figure 5.1: Filters of the first convolutional layer (`conv1_1`) from the spatial stream (VGG16 architecture) in the 2SCNN trained for action recognition on UCF101 fig. 5.1a and BEOID fig. 5.1b. For comparison, the filters from the same layer in a VGG16 network trained on ImageNet are also givenfig. 5.1c. We compute the pairwise difference between the filters in the bottom row

The filters for the temporal streams of the 2SCNN trained on UCF101 (BEOID is similar) are given in fig. 5.2. The filters are split into two columns to fit onto the page, each row represents a single 3D filter of dimensions $3 \times 3 \times 20$, the third dimension is spread out

[1] VGG16 2SCNN (UCF101), provided by Wang *et al.*, trained according to their paper on best practices in training[48], See https://github.com/yjxiong/caffe/tree/action_recog/models/action_recognition for detailed Caffe training parameters

[2] VGG16 2SCNN (BEOID), provided by Moltisanti *et al.*[28]

across the row with the first filter on the left and the last on the right. To initialise the weights of the $3 \times 3 \times 20 \times 64$ filters of the first layer in the temporal stream from the weights of the $3 \times 3 \times 1 \times 64$ filters from the ImageNet trained VGG16 model, the weights are cloned over the depth (3rd dimension). This weight initialisation helps to explain the homogeneity of the filters across the depth dimension, it is possible the filter weights get stuck in a local maximum during the training process. A small subset of filters (e.g. column 1, row 6; column 2, 5th to bottom row) demonstrate an alternating pattern of two 2D filters indicating that the network has learnt to distinguish the u and v optical flow pairs in the input as they too follow an alternating interleaving.

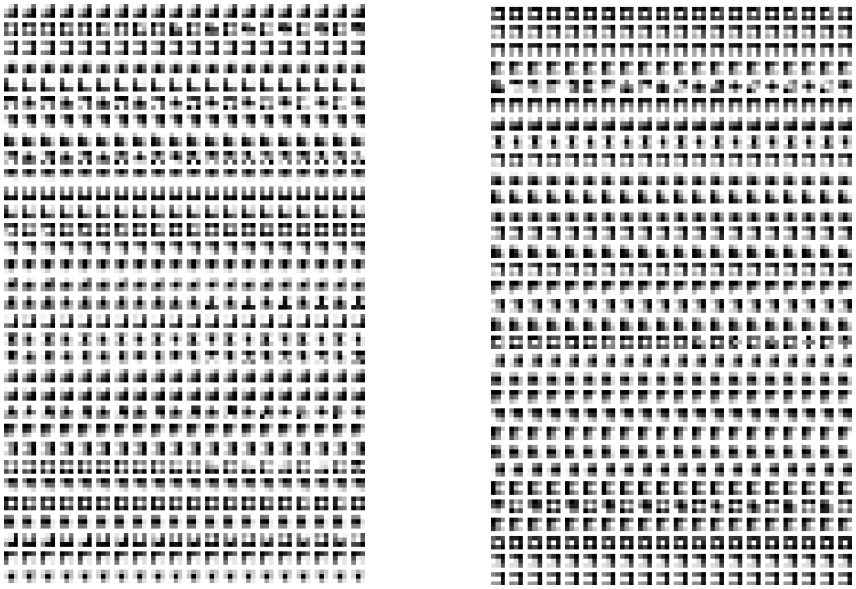


Figure 5.2: Filters from the first convolutional layer from the temporal stream in the 2SCNN trained on UCF101. Since the filters are 3D ($20 \times 3 \times 3$) we flatten the 3D tensor into a 2D image by slicing it into $1 \times 3 \times 3$ parts and joining them horizontally (so each filter spans a row).

5.3 EBP for two stream CNNs

5.3.1 Choosing a stopping layer

Two stream CNNs (2SCNN) were introduced in sec. 2.2.2, they are composed of two network streams concurrently processing the network input: the spatial stream takes a single video frame as input, and the temporal stream takes a stack of L optical flow (u, v) pairs. We produce attention maps from both the spatial and temporal stream on a per frame basis. Attention maps can be computed for the spatial stream with no modifications to EBP as only a single frame is input to the network. The temporal stream is not quite as simple since it convolves the entire optical flow input in the first layer marginalising over time; the input/output dimensions of the first layer are: $W \times H \times 224 \times 2L \to W \times H \times 64$, the layer contains $64 \times 3 \times 3$ filters, so each filter convolves over a 3D tensor of dimension $3 \times 3 \times 2L$ producing a single scalar output. If we could use EBP back to the first layer then we would be able to generate attention maps on a per frame basis for the temporal network stream, however the marginal winning probabilities become increasingly small

and sparse as the stopping layer gets closer to the first layer in the network to the point that when visualised the attention maps visually provide little information as can be seen in fig. 5.3. Stopping at any other layer above the input provides only a single attention map so we have to use a different approach to generate attention maps for each frame with EBP.

Figure 5.3: The affect of stopping EBP at different layers (UCF101 boxing)

Selecting the stopping layer for EBP was an exercise in trial and error, we computed attention maps by stopping at various layers in the network and found that the third pooling layer provided a good compromise between visual interpretability and resolution of attention (i.e. the size of the area for which the marginal winning probability applies). At the third pooling layer of VGG16, the attention map dimensions are 28×28 and so each marginal winning probability covers a $224/28 \times 224/28 = 8 \times 8$ patch of pixels in input space giving acceptable spatial resolution. See fig. 5.3 for a visual comparison of attention maps computed for the same frame using different stopping layers.

5.3.2 EBP for the temporal stream

We propose a novel method for generating attention maps on a per frame basis for temporal streams in the 2SCNN architecture using EBP. For a temporal stream network with temporal extent L, a window W_τ over $L+1$ video frames is constructed such that the first frame of the window has index τ hence the window covers frames τ to $\tau + L + 1$. The temporal stream input T_τ corresponding to the frames in the window W_τ is computed to produce a stack of optical flow frames of size $2L$. We compute a forward pass and a backward pass using EBP to generate an attention map A_τ corresponding to the window W_τ. The window is then slid along by a single frame to produce a new window $W_{\tau+1}$ and the process is repeated to produce another attention map $A_{\tau+1}$. The sliding window is initialised at $\tau = 1$ (the first frame). For a video f frames long, we are able to produce $f - (L+1)$ attention maps as there are insufficient frames from frames with indices $\tau > f - (L+1)$ to form a full input to the temporal network hence we cannot compute an attention map. A graphical depiction of this process is presented in fig. 5.4

The method produces attention maps for windows of frames but can't give us a single frame timestep level resolution since the attention map applies equally to all frames in the window. It is an arbitrary choice which spatial frame we associate with A_τ providing it is in $[\tau \,..\, \tau + L + 1]$ (i.e. in the associated window). Several obvious choices come to mind: the first frame τ, the middle frame $\tau + (L+1)/2$ and the final frame $\tau + L + 1$. To evaluate which of these makes the most sense we overlaid the attention map A_τ on the chosen frame τ_{underlay} and recombined the overlaid frames into a video. The videos illustrate the impact of the frame choice:

- $\tau_{\text{underlay}} = \tau$: The attention map indicates the salient regions in the next $L+1$

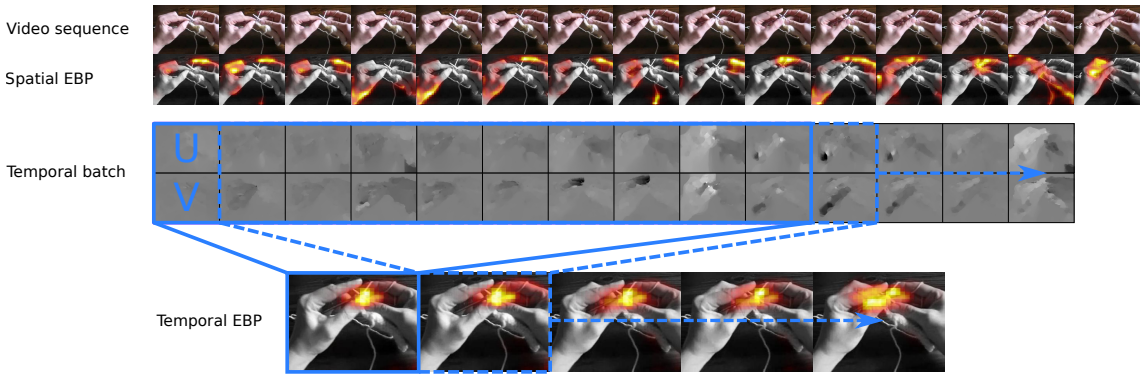


Figure 5.4: Sliding window technique generating attention maps on a per frame basis, example video clip from UCF101

frames.

- $\tau_{\text{underlay}} = \lfloor \tau + (L+1)/2 \rfloor$: The attention map indicates the salient regions of the last $\lfloor (L+1)/2 \rfloor$ and future $\lceil (L+1)/2 \rceil$ frames.
- $\tau_{\text{underlay}} = \tau + L + 1$: The attention map indicates the salient regions over the last $L+1$ frames.

We visually assess which frame to use (out of the 11 frames used to calculate the optical flow in the sliding window[3]) to overlap the temporal attention map onto. We take a short clip of the break in a billiards game and show how the choice of the frame to underlay the attention map affects the interpretability of the output in fig. 5.5. The figure shows attention maps from six sliding windows, highlighting the first of the six sliding windows in dotted pink border and the last in dotted green border. We assess three different choices of the underlaying frame, particularly the first frame in the batch ($t = 0$) at the top row, the middle frame in the batch ($t = 5$) at the middle row, and the last frame ($t = 10$) at the bottom. The figure shows that overlays on both ($t = 0$) and ($t = 5$) highlight the future trajectory of the ball rather than where the action is taking place. However, underlaying the attention map on the last frame ($t = 10$) is the most easily interpretable as the attention map localises the past and current path of the ball, rather than the future motion. From this and other examples, we conclude that using the last frame in the batch, namely $t = 10$, better highlights salient parts of the motion and offers the best visualisation of temporal attention maps. We use this choice in all overlays in this thesis.

[3]The temporal network has a batch size of L formed from optical flow frames each of which is derived from a pair of consecutive frames hence the attention maps apply to the $L+1$ spatial frames used to generate the optical flow frames.

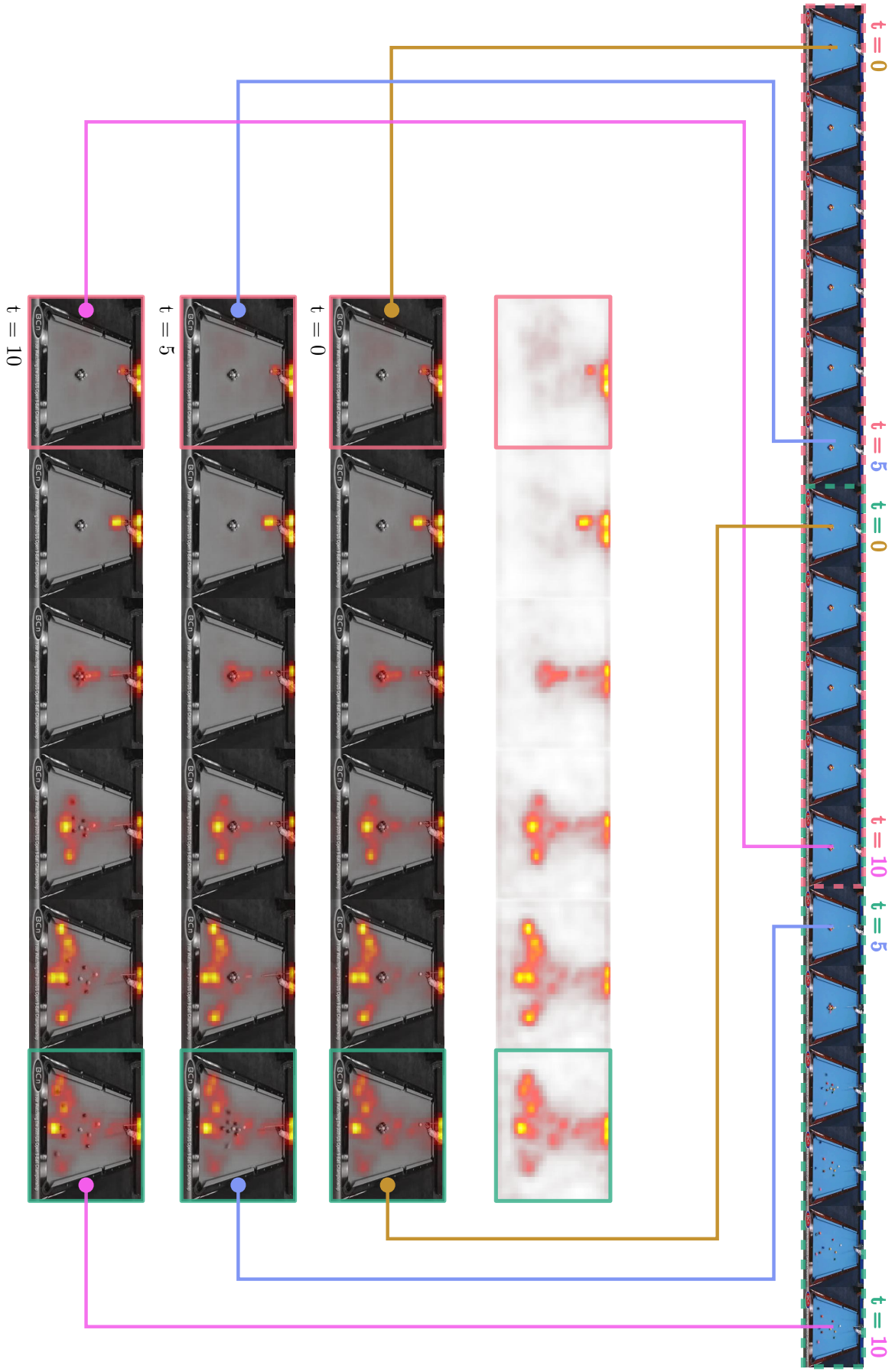


Figure 5.5: Selecting a frame from the input batch to underlay the temporal attention map (*Billiards*, UCF101)

5.4 EBP Attention map evaluation

In this section we qualitatively and quantitatively assess the generated attention maps for the spatial and temporal networks using both contrastive and non-contrastive EBP. We assess the *jitter* of the attention map sequences across both trained models (UCF101 and BEOID) using a metric we propose called *L2-jitter*. In addition to jitter, we also assess the accuracy of the attention maps on the BEOID trained network by comparing the peak of the attention map and action location. Finally we present selected attention map sequences demonstrating features present across the set of generated maps.

We use the videos from the test set used to evaluate the accuracy of the network (see table 5.1).

We make use of the following abbreviations in this section:

- **SC**: Attention maps for Spatial network generated from Contrastive EBP
- **SNC**: Attention maps for Spatial network generated from Non-Contrastive EBP
- **TC**: Attention maps for Temporal network generated from Contrastive EBP
- **TNC**: Attention maps for Temporal network generated from Non-Contrastive EBP

5.4.1 Network details

Table 5.1: VGG-16 Network stream accuracy on BEOID and UCF101.

Dataset	Stream	Accuracy
BEOID	Spatial	83.9%
	Temporal	92.9%
	Post convolution[4] fusion	94.8%
UCF101	Spatial	78.4%
	Temporal	87.0%
	Late fusion	91.4%

Videos clips were decomposed into constituent frames and encoded as 8-bit integers using JPEG compression. BEOID video is recorded at 640×480 resolution, UCF101 at 320×240[5].

Both network streams were trained starting with network weights taken from a network with the same architecture trained on ImageNet, and so we inherit a lot of the hyperparameters common to ImageNet networks: Namely a feature scale of $[0, 255]$, mean subtraction of $(103.9, 116.8, 123.7)$ and channel order: BGR.

The spatial stream takes a single color video frame input I_τ of dimensions $224 \times 224 \times 3$ in BGR format in the range $[0, 255]$ mean centred about the value $(103.9, 116.8, 123.7)$.

The temporal stream takes a stack of optical flow frames specified by a starting frame index τ of duration L in frames ($L = 10$ for our experiments). fig. 2.15 gives an accompanying graphical depiction of how frames are stacked for input to both network streams. The

[4] Post convolution fusion refers to the combination of the network streams after the convolutional layers (before the fully connected layers) combining the two streams into a single spatio-temporal stream. This idea was proposed by Feichtenhofer *et al.* in [10].

[5] UCF101 is collected from YouTube so it possible that videos are upsampled to the desired resolution.

optical flow V_τ of a pair of frames I_τ and $I_{\tau+1}$ with respective frame indices τ and $\tau+1$ are obtained by using the TVL1[51] optical flow estimation algorithm. The resulting motion vectors can be positive or negative, but are recorded in image form. To handle negative values in the stored optical flow, the flow is rescaled to be in the range $[0\,..\,254]$ where 127 represents 0, anything below 127 is negative and anything above is positive. Similarly to the video frames these were stored as 8-bit integers using JPEG compression. Optical flow is computed in both u and v directions so there are $2L$ optical flow frames in the input to the temporal stream, they are stacked such that frames with even offsets from τ are in the u direction and in the v direction for odd offsets. Let T_τ be the input to the stream, and T_τ^k be the optical flow frame in the input at offset k, then the full input T_τ is defined as $T_\tau^{2k} = V_{\tau+2k}^u$ and $T_\tau^{2k+1} = V_{\tau+2k+1}^v$ for $k \in [0\,..\,L-1]$. Once the optical flow is read back from JPEG format stored on disk into memory, they are transformed to be in the range $[-127.5, 127.5]$ performed by mean centring around 127.5.

5.4.2 Evaluating attention maps for jitter

Contrastive attention maps (those produced by contrastive EBP) demonstrate large variances between consecutive maps where there is little change in the corresponding video frames. We expect the attention maps to change proportionally to the change in the corresponding video frames; we call this property of attention map sequences *jitter*. Attention map sequences with low jitter are those in which the salient regions in one map are also salient in the following map. High jitter sequences can be considered unstable: regions flip between salience and irrelevance across the sequence. fig. 5.6 shows a cliff diving clip in which the spatial non-contrastive attention maps are considered to have low jitter, they localise the diver through each consecutive frame. The spatial contrastive attention map sequence is one we consider to have high jitter; specifically the first frame correctly localises the diver, but then in the next frame does not localise the diver instead highlighting an irrelevant region in the top right. Frame 4 highlights the diving platform but neither frames 3 or 5 do. In contrast, the temporal non contrastive attention maps have low jitter, they localise the action well and have significant overlap between pairs of consecutive attention maps. The temporal contrastive attention maps have little overlap frame to frame with salient regions appearing and disappearing between frames and so are declared to suffer from high jitter.

We quantitatively assess the jitter of a sequence of attention maps by first computing the jitter between pairs of consecutive frames, then averaging the jitter between pairs over the whole sequence to give a jitter score for each video clip. We quantify jitter by means of a metric we call *L2-jitter*. We compute the L2-jitter between pairs of consecutive attention maps by computing the L2 element-wise difference and summing over the element differences to produce a scalar score per attention map pair. See sec. 8.2 for a table of clips listing the videos with extreme jitter values in the datasets.

We compare the distribution of average L2-jitter per clip for each network stream and EBP type in fig. 5.7. The number of clips and average frame counts used for this analysis are detailed in tbl. 5.2. The distribution for L2-jitter in UCF101 (fig. 5.7a) highlights the large disparity between contrastive and non-contrastive EBP in the spatial stream but fails to capture this on BEOID, or in the temporal attention maps. Over the set of tested clips from BEOID, the distribution of L2-jitter for the spatial contrastive attention maps does not differ significantly from the spatial non-contrastive attention maps (fig. 5.7b), however

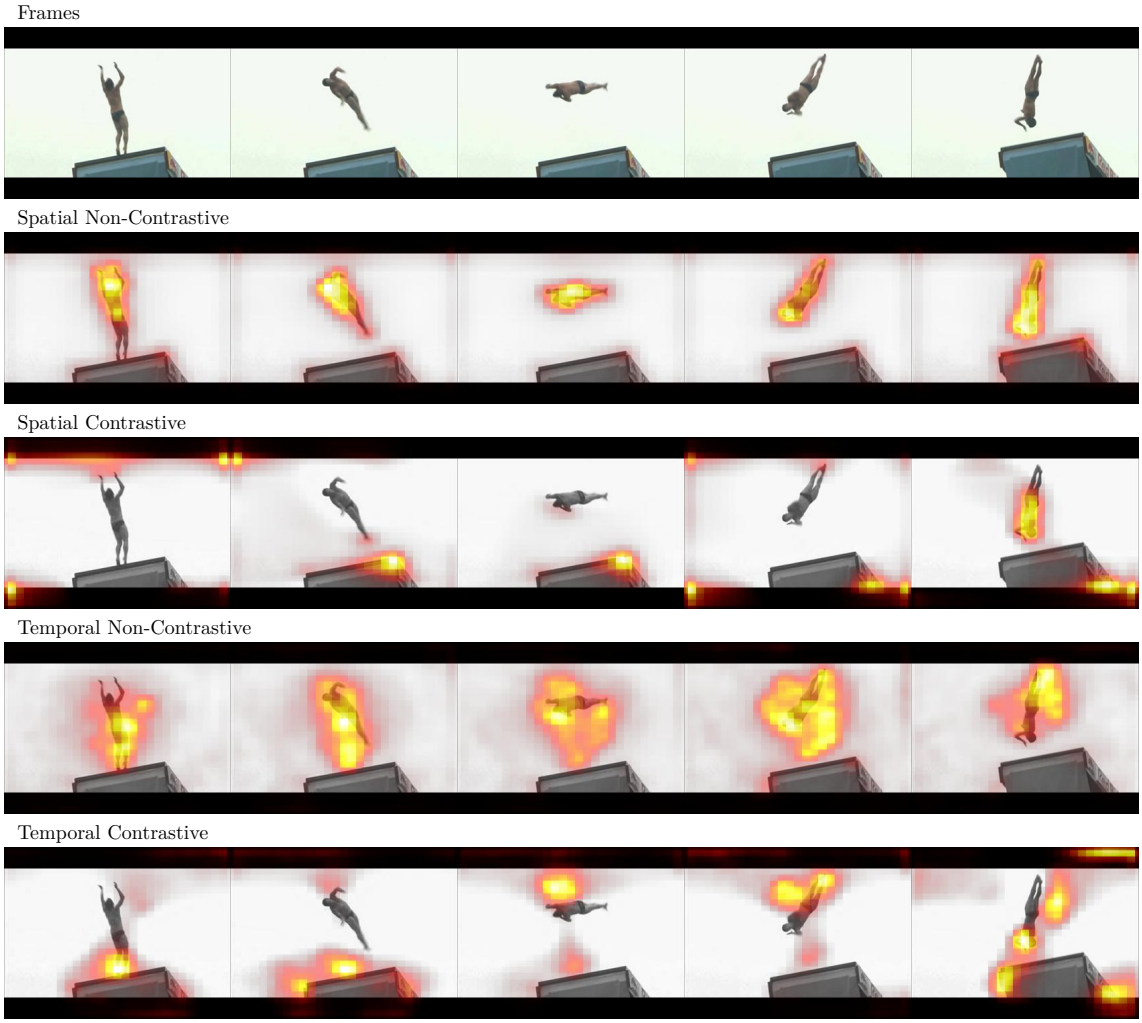


Figure 5.6: Attention map sequences with high and low jitter: The contrastive attention maps suffer significantly higher jitter than the non contrastive attention maps for both the temporal and spatial streams (Cliff diving, UCF101)

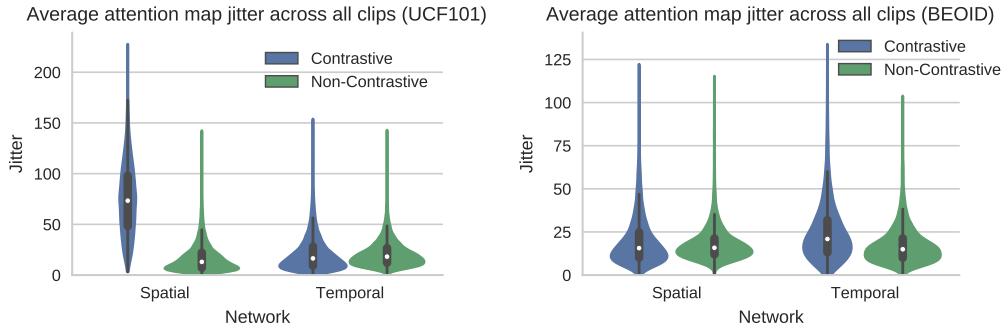
our qualitative analysis contradicts this; we observe higher levels of jitter in the spatial contrastive attention maps suggesting the need for a better metric.

The BEOID video dataset is shot from a head mounted gaze tracker, so all videos have at least some camera motion, whereas UCF101 is composed mostly of static camera shots. For attention map sequences with camera motion, we observe similar L2-jitter results for sequences qualitatively considered to have low jitter and high jitter; the measure fails to quantify this visual difference in clips with large camera movements due to producing large L2-jitter values for similar attention maps that are shifted across the frame. Our L2-jitter analysis is more appropriate for videos shot from static cameras. We suggest an alternative metric, the *earth movers distance* (EMD) to better cope with camera movement. The EMD considers a 2D array as piles of earth on a surface, the distance between the two piles of earth is computed as the minimum effort required to shift earth such that the first array is transformed into the second one.

We further subdivide the L2-jitter distribution by location for BEOID in fig. 5.8 to determine whether jitter varies by location, however the plot shows that there is little change between location.

Table 5.2: Test dataset summary statistics

Dataset	Fold	Clip Count	Average frame count per clip	Total number of frames in dataset
UCF101	1	100	186	18573
BEOID	1	155	47	7294



(a) UCF101

(b) BEOID

Figure 5.7: Distribution of average jitter over clips for each network stream and EBP type.

We believe the inferiority of the attention maps produced by contrastive EBP compared to non contrastive EBP is due to feature overlap between classes. Contrastive EBP aims to produce an attention map highlighting the region that discriminates between classes unlike non-contrastive EBP which highlights regions that contribute to the activation of the class neuron regardless of whether they are discriminative. Contrastive EBP produces discriminative attention maps by computing two attention maps at the target stopping layer L_{stop}. The first attention map A_{pos} is that computed by non-contrastive EBP. The second A_{neg} is computed by first inverting the weights in the starting layer to produce a dual unit for each neuron recognising the negation of the original class, e.g. a 'press-button' class neuron

has a dual neuron 'not-press-button'. The attention maps are combined and thresholded at 0 to produce the contrastive attention map: $A_{\text{contrastive}} = \max(A_{\text{pos}} - A_{\text{neg}}, 0)$. This contrastive attention map will have non-zero components where the attention in A_{pos} is greater than A_{negative}. The jitter in the contrastive attention maps we generate can be explained by very similar A_{pos} and A_{neg} where regions with similar levels of attention fluctuate between being present in the contrastive attention map $A_{\text{contrastive}}$ when the region has higher attention in A_{pos} than A_{neg}, and disappearing when A_{neg} has higher attention than A_{pos}. The high prevalence of jitter across the dataset suggests that there is significant feature overlap in classes causing high similarity between A_{pos} and A_{neg}.

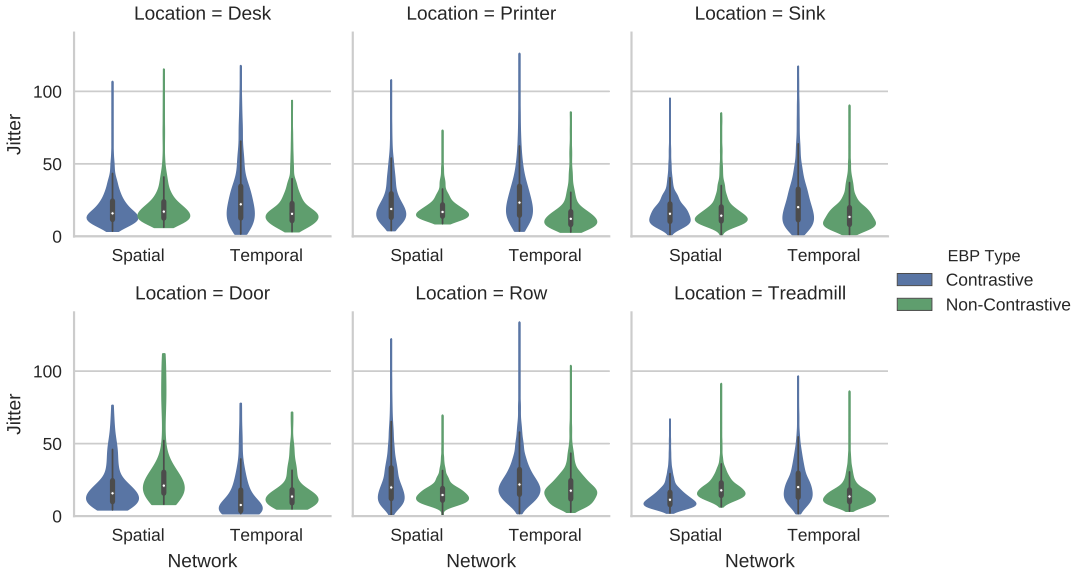


Figure 5.8: Distribution of average jitter over clips broken down by location.

5.4.3 Evaluating attention map quality by egocentric gaze

In this section we evaluate the quality attention maps generated in the BEOID dataset by comparing the action location to the attention map peak location, smaller distances between the peak and action location suggest the network has learnt to recognise the action rather than the surrounding context.

The BEOID dataset is provided with gaze data for each video: the operator performing the action is wearing a head mounted video camera and gaze tracker that both records the operator field of view and the point of gaze across the recorded 2D video frame.

Human gaze flips between two modes of operation: fixation and saccading. When fixating, the eye is stationary focusing on a specific object in the field of view. Saccades occur between periods of fixation; during the saccade the eye darts around the field of view.

We use the gaze data recorded from a head mounted gaze tracker worn by the operator in the BEOID dataset as a proxy variable for the action location as the two are correlated[22]; a person's gaze when performing an action is directed towards the action. We compare the location of the attention map maximum (i.e. the most salient region across the map) to the location of the operator's gaze during periods of fixation to comparatively assess the

different EBP methods over both network streams. We first compute the location of the attention map peak and scale it by the dimensions of the source frame, then compute the L2 distance in pixels between the scaled peak location and the gaze location (recorded as a pixel location on the video frame). A graphical depiction of this measurement is given in fig. 5.9.

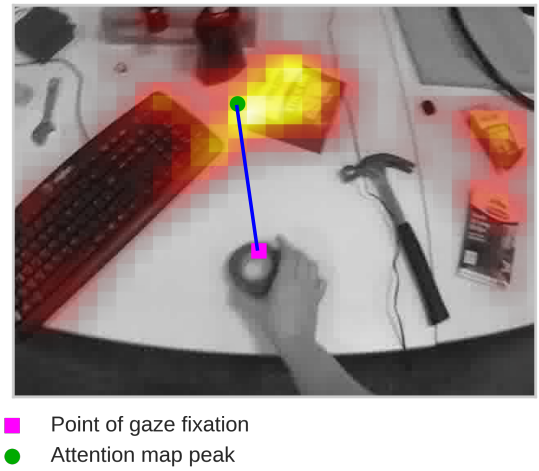
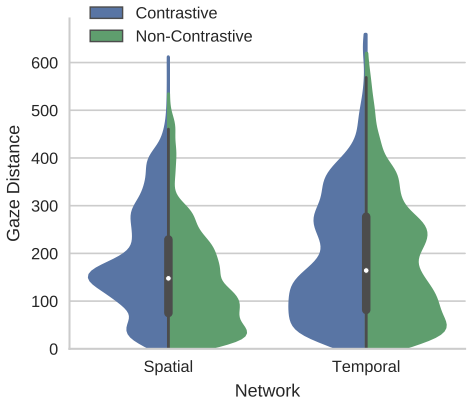
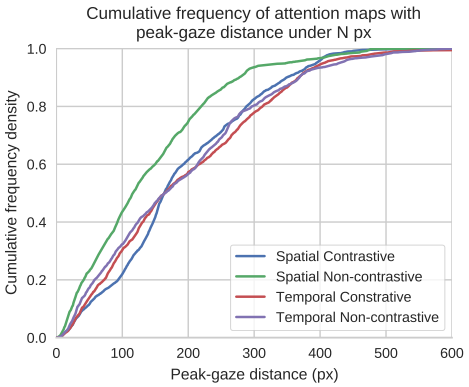


Figure 5.9: Attention map peak to gaze is measured in pixels (px) along the blue line from the point of fixation of the operator (pink) to the attention map peak (green)

We perform the comparison over all attention maps for which corresponding gaze data is available, filtering out attention maps corresponding to frames in which the operator is not fixating. We present the distribution of distances for each network stream and EBP type in fig. 5.10a and record the proportion of attention maps under a certain distance threshold in fig. 5.10b. We give a further breakdown of distances by action location (the environment in which the video was taken) in fig. 5.11. Spatial non-contrastive attention maps are consistently superior to spatial contrastive in peak-gaze distance error. We note that temporal contrastive and non contrastive are fairly consistent in peak-gaze distance error. The number of attention maps used for this analysis is given in fig. 5.12.



(a) Distribution of attention map peak - gaze distance across all attention maps



(b) Proportion of attention maps with peak-gaze distance under given threshold

Figure 5.10: Attention map peak - gaze distance plots

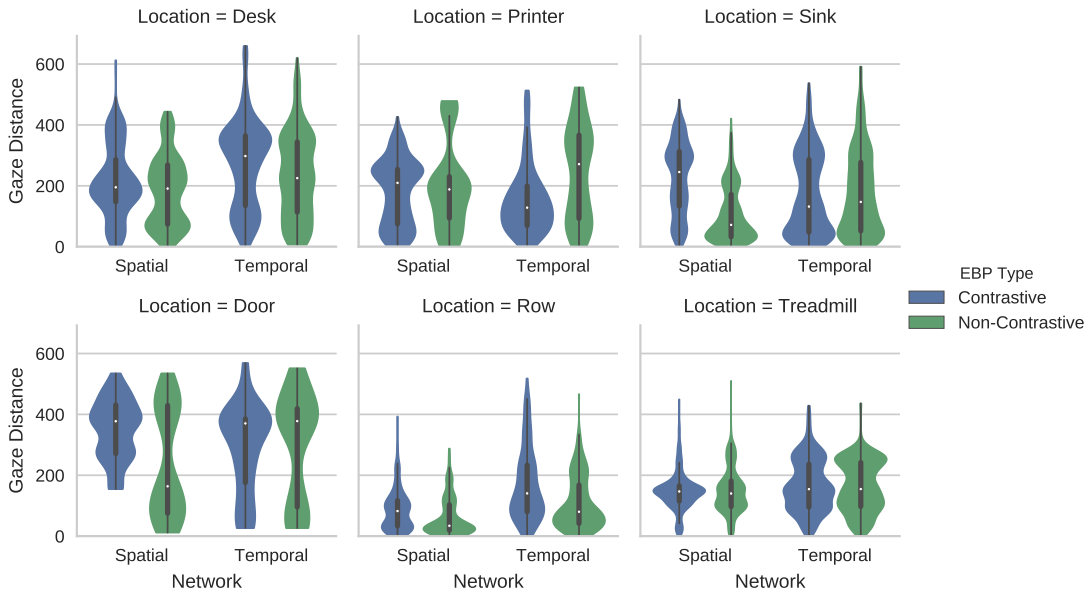


Figure 5.11: Average attention map peak - gaze distance per clip across all locations (BEOID)

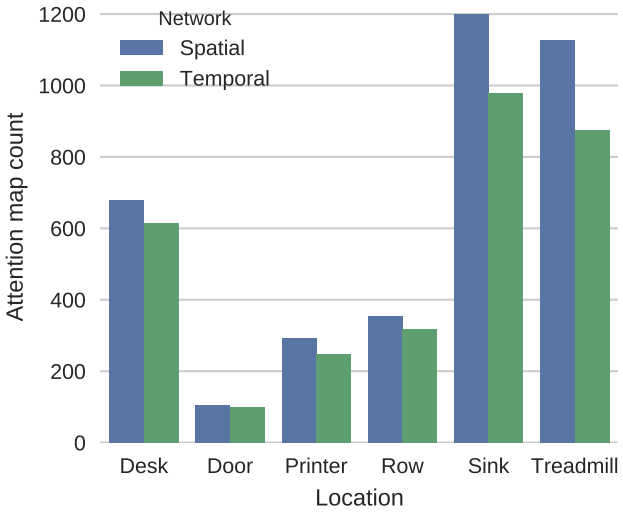


Figure 5.12: Attention map with gaze counts: Temporal has fewer attention maps due to the limited number of temporal window positions

5.4.4 Qualitative attention map evaluation

We give a qualitative analysis of selected attention map sequences to determine features learnt by the two streams for different action classes. We also make note of pathological behaviour where present and provide plausible explanations. We first examine examples from UCF101 and then BEOID concluding with general comments on features observed from both datasets.

UCF101

We select nine examples of different actions from the UCF101 dataset: *mixing*, *writing on board*, *tennis swing*, *playing flute*, *soccer penalty*, *rope climbing*, *hammering*, *playing guitar* and *swinging*; for which we infer the features recognised by the network streams. We note pathological behaviour where present and supply possible explanations. For each example we give the ground truth class following the figure reference.

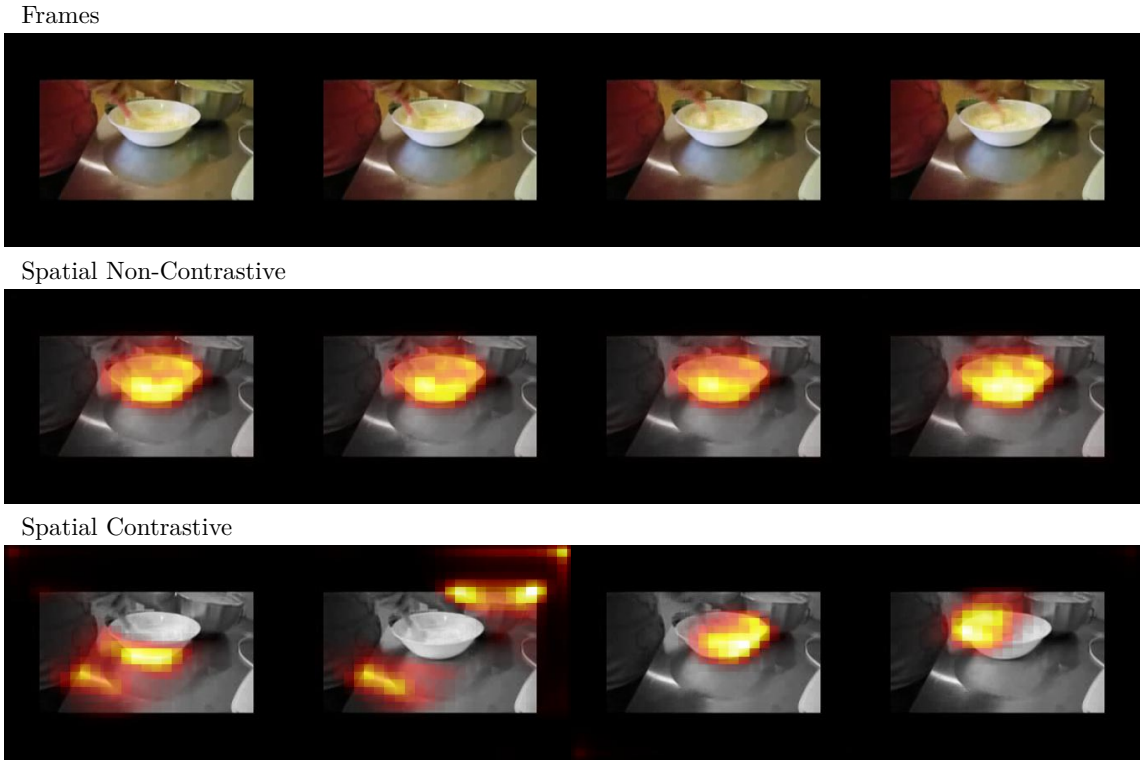


Figure 5.13: UCF101 Example: `v_Mixing_g01_c04`

Fig. 5.13 (mixing): Good SNC maps, the attention is localised to the bowl and is consistent across the frame sequence. SC suffers from heavy jitter, regions to which attention is localised are not consistent across the frame sequence, they jump around the entire frame with little overlap despite the similarity between frames; frames 3 and 4 localise attention to the bowl. The localisation of attention to regions over the bowl indicate that the spatial stream has learnt to recognise bowls as a proxy for mixing since mixing most often occurs in bowls.

Fig. 5.14 (writing on board): A good example of where the spatial network has learnt features sufficient for distinction between classes, but not corresponding to the action. The

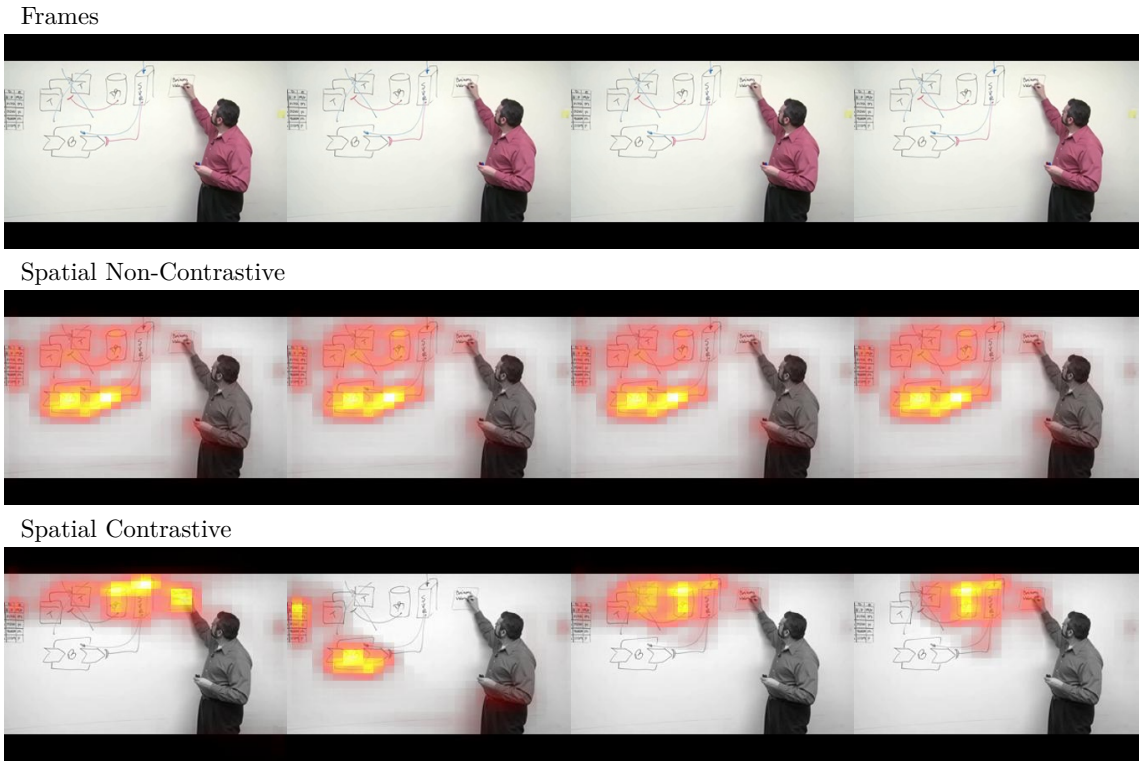


Figure 5.14: UCF101 Example: `v_WritingOnBoard_g04_c03`

whiteboard contains writing over which most of the attention is distributed for both SC and SNC. If we consider the region localising the action of writing on the board to be the region covering the hand and pen and a small surrounding patch of the whiteboard, then this comprises a very small proportion of the whole frame, it is much easier for the network to recognise the large patches of writing on a white background than it is to recognise this small patch in which the region actually takes place; the network is has not learnt to recognise the action for this class in this example, but instead uses proxy features (patches of writing on the board) for differentiating between classes.

Fig. 5.15 (tennis): Noisy attention maps for both SNC and SC. A good proportion of the attention is distributed the regions covering the tennis player and his racket in the SNC maps, SC suffers from jitter as most regions of attention in one frame are not present in the next frame. Attention is distributed to the fence suggesting that there are multiple examples in the training dataset in which fences are present in the `TennisSwing` action.

Fig. 5.16 (flute): Excellent attention maps for SNC, TC, TC with accurately localised attention. The SNC maps localise attention to the flute, and face and arms of the player consistently across the frame sequence. The SC maps suffer from jitter but the localised regions are generally considered to be relevant the `PlayingFlute` action. The TNC maps are similar to those generated by SNC indicating agreement between the two streams in relevant features. The optical flow for the sequence outlines the player so it is probable that the temporal network is acting similarly in the spatial network: recognising by appearance rather than by motion. The TC maps localise attention to the face and tip of the flute unlike the TNC maps which localise over the full length flute.

Fig. 5.17 (soccer penalty): The SNC maps are consistent and correctly localise the player,

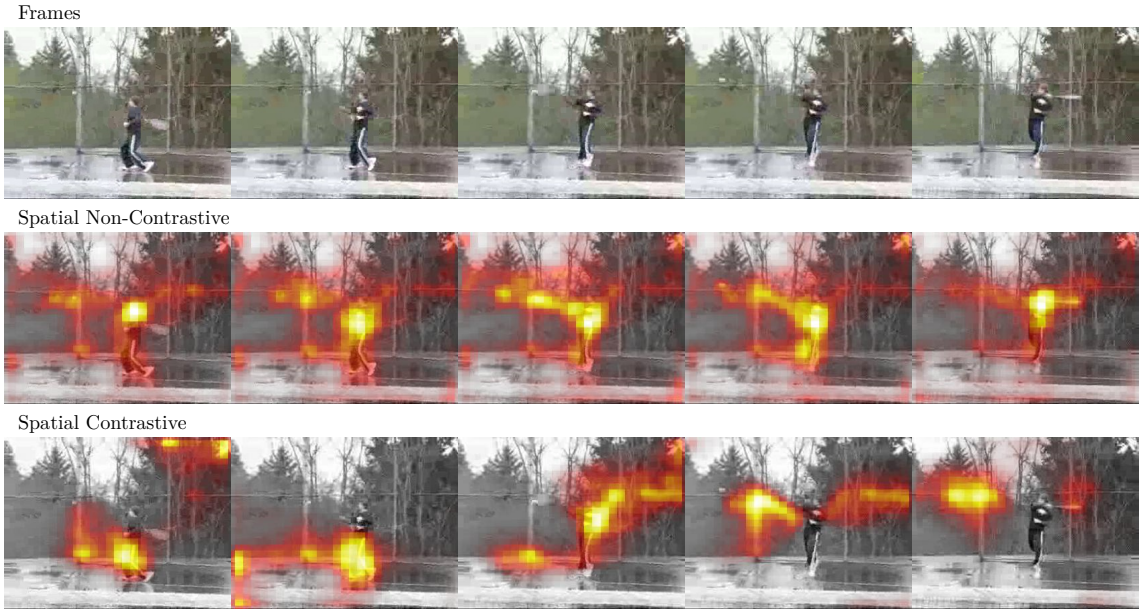


Figure 5.15: UCF101 Example: `v_TennisSwing_g07_c02` (high spatial contrastive)

Figure 5.16: UCF101 Example: `v_PlayingFlute_g03_c05`

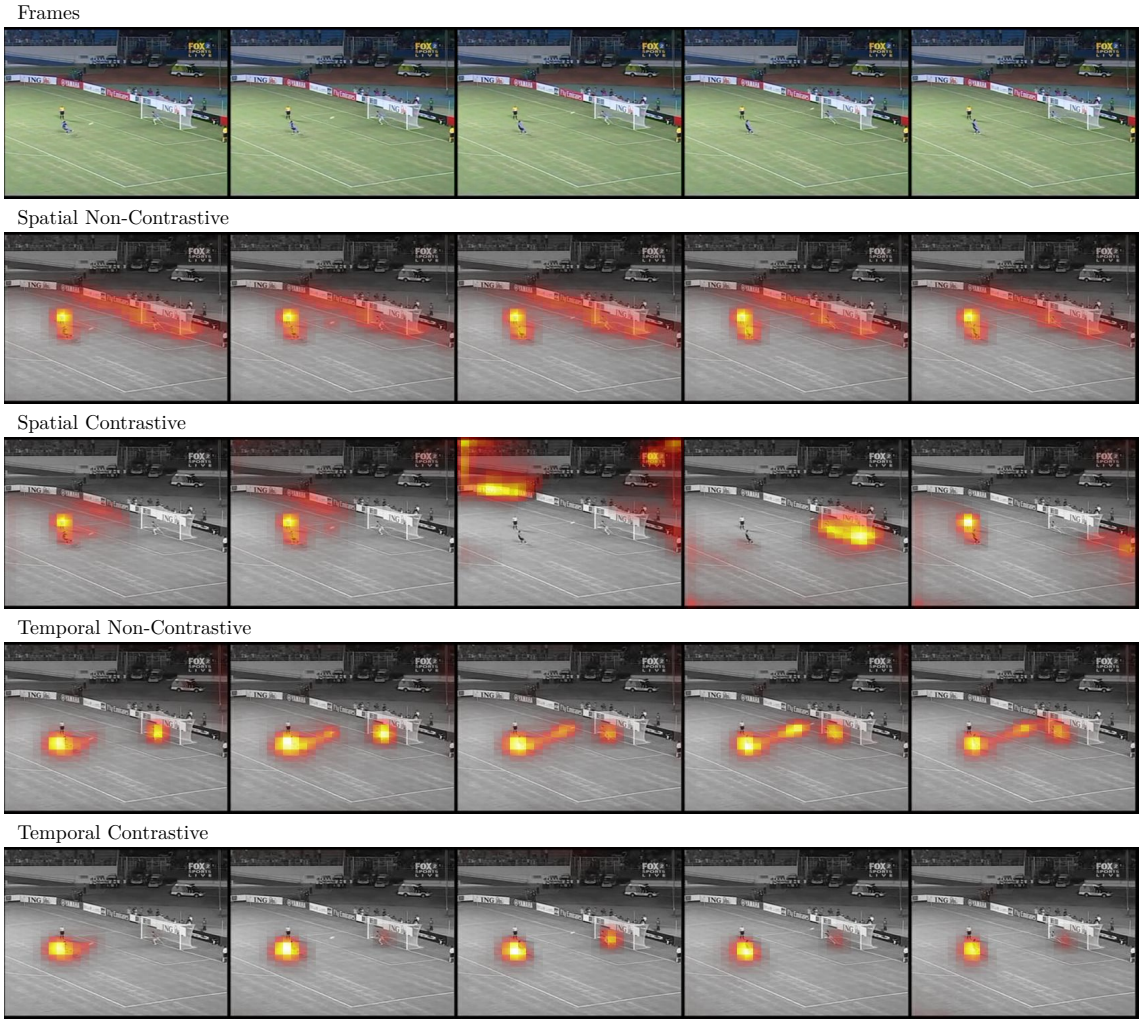


Figure 5.17: UCF101 Example: `v_SoccerPenalty_g01_c06`

referee, and goal goalkeeper; attention is also distributed to the surrounding advertisements on the pitch walls, this could be due to `SoccerPenalty` being the only class conducted in a large stadium with low-zoom footage as it would be a positive discriminator for the class. The SC maps suffer from jitter, but localise the player and referee in 3/5 of the frames, the goal keeper is only localised in 1 frame. The TNC maps localise the action to the player, ball and goalkeeper, the moving objects comprising the action. The trail of attention behind the ball is due to the temporal window over which the attention maps are calculated for the temporal network. The TC maps localise the player and goal keeper but not the ball, we posit this is due to the overlap in classes featuring moving balls, like `TennisSwing`, `Billiards`, the contrastive method will cancel out common winner neurons resulting in the attention not being distributed to regions over the moving ball.

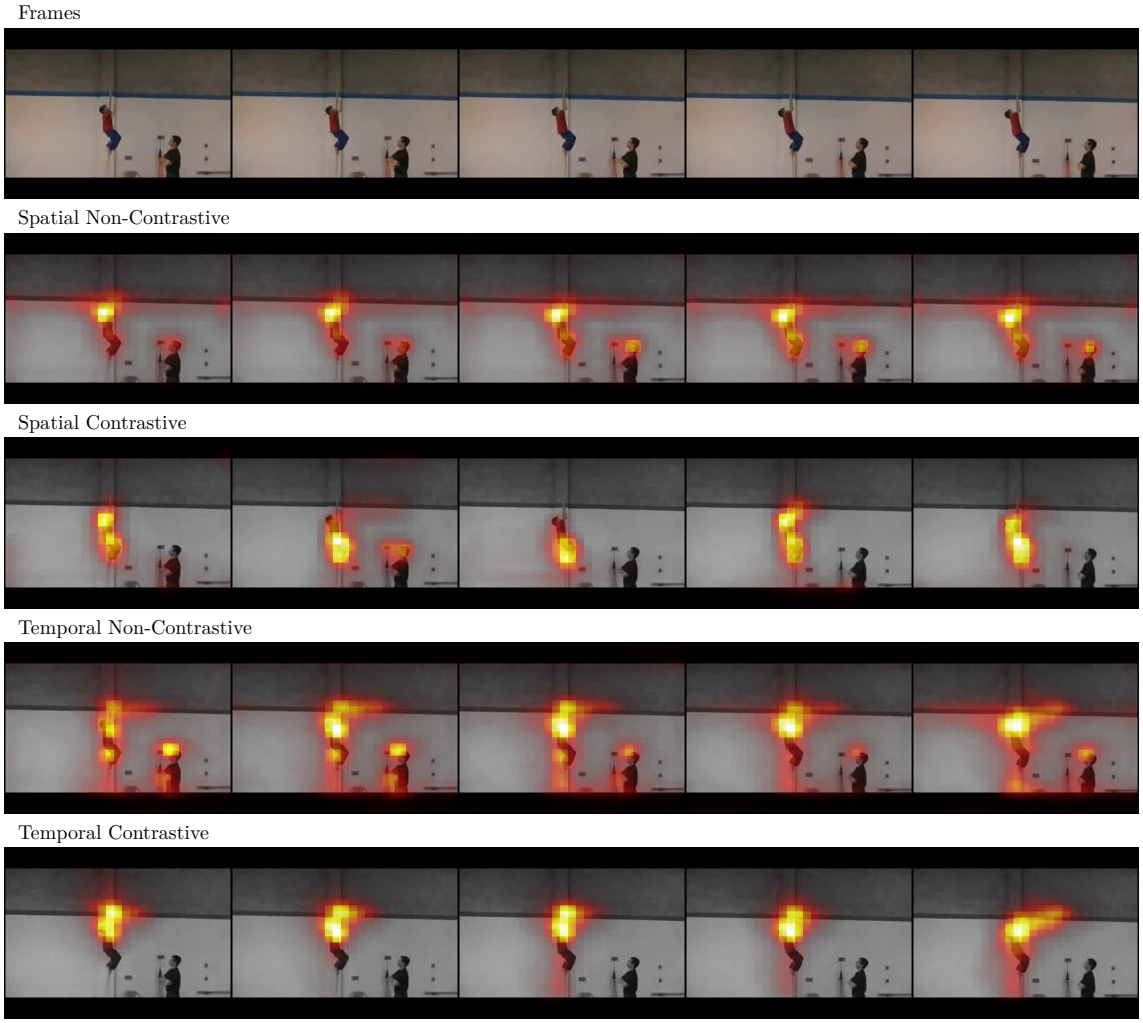


Figure 5.18: UCF101 Example: `v_RopeClimbing_g05_c07`

Fig. 5.18 (rope climbing): This is one of the few examples in which SC performs better than SNC at localising the action, SC distributes attention to the climber and the rope in all but frame 2 in which the instructor's face is localised, unlike SNC in which the instructor is localised throughout the frames. TNC and TC exhibit similar differences to SNC and SC, the instructor is localised in TNC, but not in TC. The localisation of the climber is finer in TC, but only covers the upper portion of the body whereas TNC covers the full body of the climber in addition to the rope below.

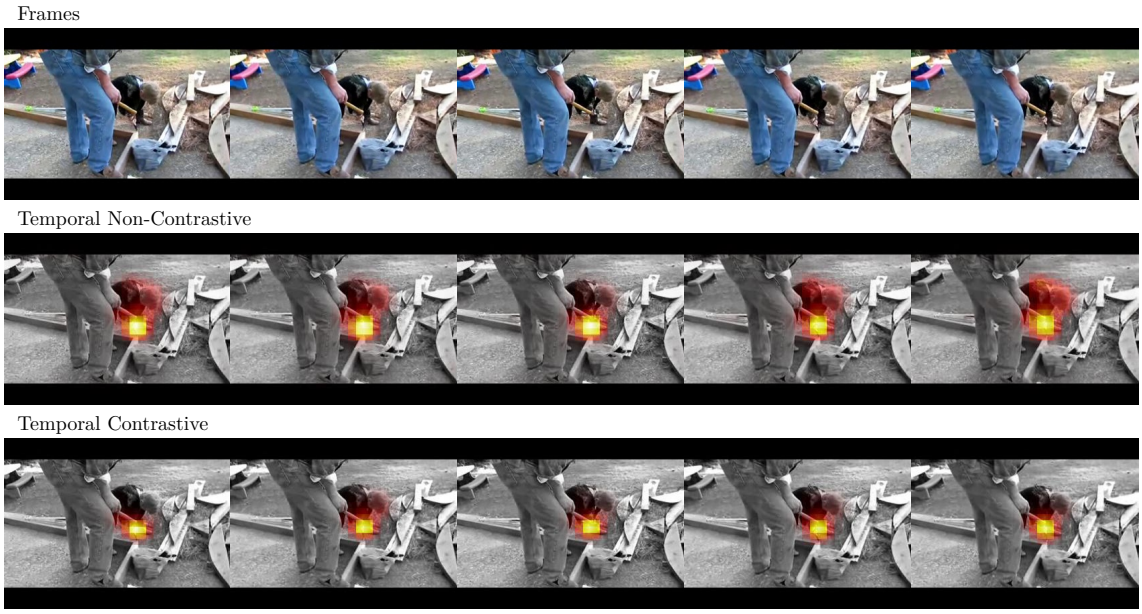


Figure 5.19: UCF101 Example: `v_Hammering_g07_c05`

Fig. 5.19 (hammering): TC localises hammer action better than TNC, both are good restricting attention localisation to the child and hammer, but TC further restricts the attention to the region in which the hammer is moving suggesting the temporal network has learnt to recognise the repetitive motion of hammering.

Fig. 5.20 (playing guitar): TC fails to localise the strumming motion, instead localising the top of the musicians head. The TNC attention maps spread over a significant proportion of the frame, but do indeed localise the musician and peak around the soundhole and fretboard.

Fig. 5.21 (swing): The features highlighted in the attention maps for both TNC and TC are hard to determine, attention is localised to the reflection of light on the child's hair which will exhibit a strong swinging motion in the optical flow frames, the temporal network may have learnt a feature encapsulating objects following this swinging motion.

Figure 5.20: UCF101 Example: `v_PlayingGuitar_g05_c01`

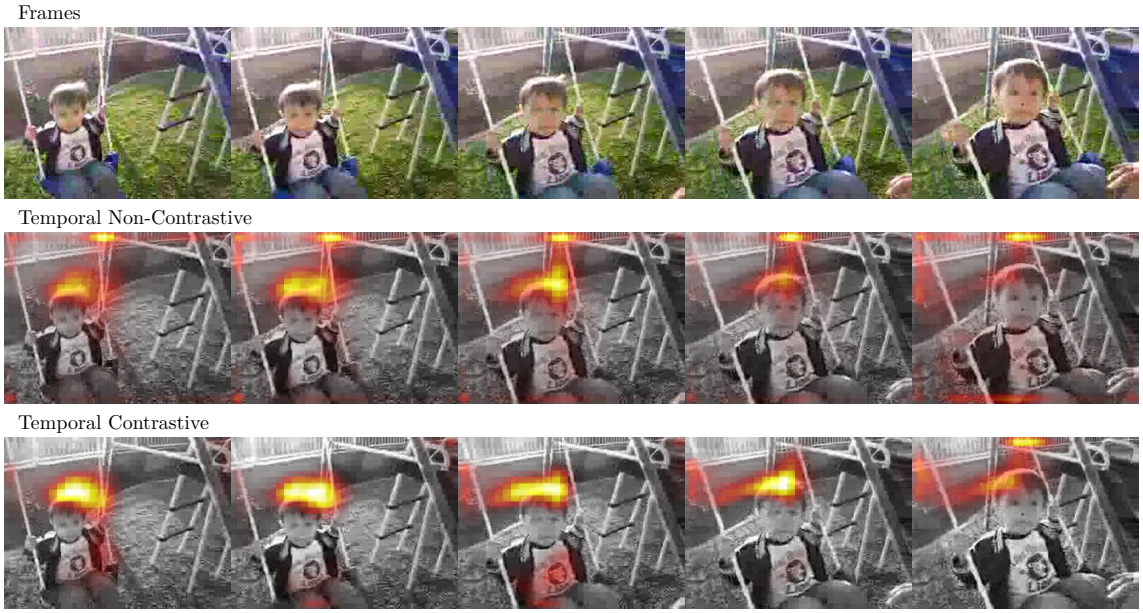


Figure 5.21: UCF101 Example: `v_Swing_g06_c07`

BEOID

We present four actions from BEOID, these are: *open door*, *press button*, *turn tap* and *stir spoon*. For the *press button* action we show three examples at different locations and compare the attention maps across the location. The examples are selected to make conclusions of interest in understanding the features learnt by the networks trained on BEOID. The examples chosen are representative of the full gamut of attention maps for BEOID. For each example we note the action and location of the example after the figure reference as (*action*, location).

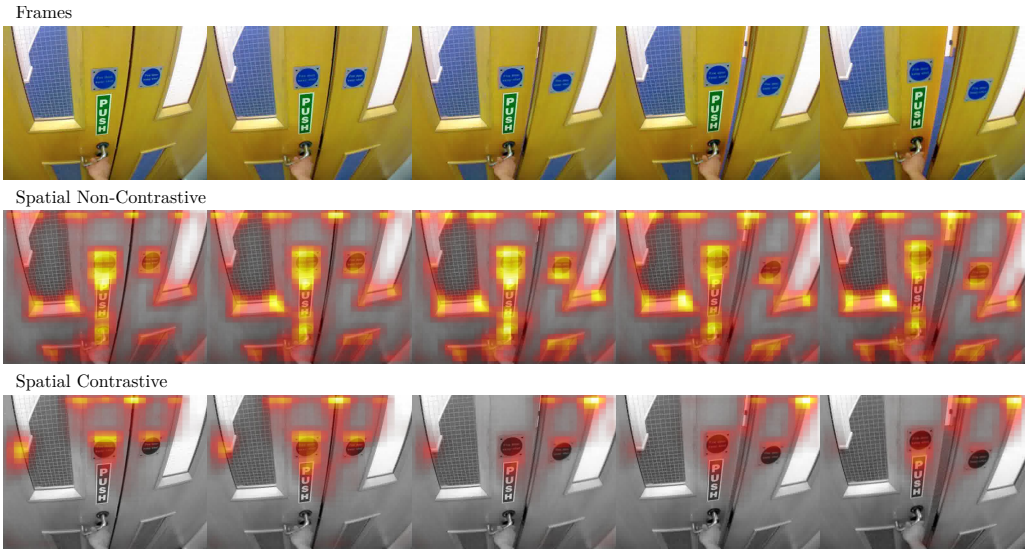


Figure 5.22: BEOID Example: `04_Door2_open_door_284-333`

Fig. 5.22 (*open door*, door): All door videos are shot in this location with this specific door. The SNC maps indicate that the salient regions are the edges of the windows, the badges above the handle, and the handle itself suggesting that the network has learnt to recognise the appearance of the door (probably overfitting to the appearance of the specific door pictured) in addition to the hand on the handle to discriminate this action. The SC maps demonstrate less jitter than compared to other actions in the dataset although they do not localise the hand on the door handle. No other clips for different action classes feature the presence of a door, so the appearance of a door sufficient for distinction between other classes.

Fig. 5.23 (*press button*, treadmill): The only action recorded on the treadmill is press button, thus recognising the treadmill should be sufficient to discriminate this action from other press-button actions conducted with different objects. The red stop button of the treadmill is a good indicator that the object of interaction is the treadmill as it doesn't appear on other exercise equipment in the other action classes. The SNC maps have low jitter, primarily localising the hand and the red stop button of the treadmill suggesting that the spatial stream is both recognising the object of interaction, the treadmill dashboard, and the shape of the hand in the interaction. The SNC maps are consistent across the frames and primarily localise the text on the display and the surrounding border of the display. It is noteworthy that the red stop button is not localised in the SC maps despite not being present on other exercise machines recorded in the dataset, the closest similar object in appearance is the red tap in the *sink* location.

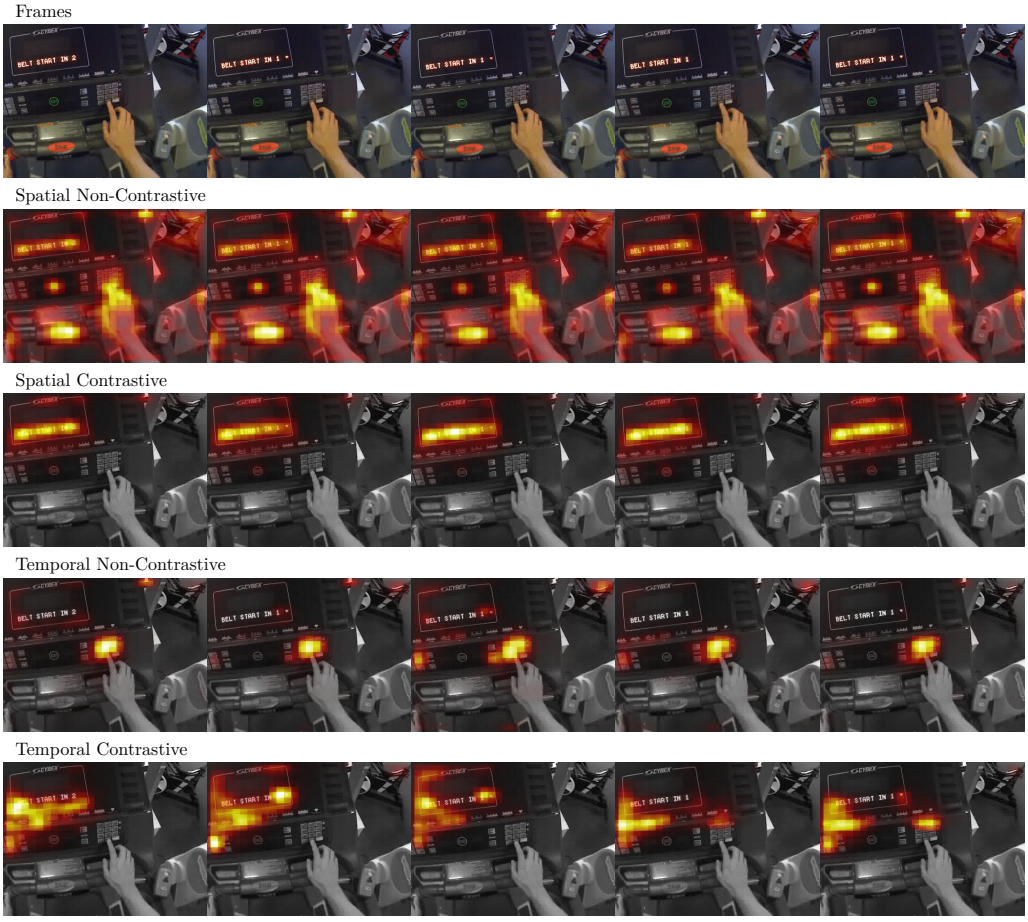


Figure 5.23: BEOID Example: `07_Treadmill1_press_button_193-305`

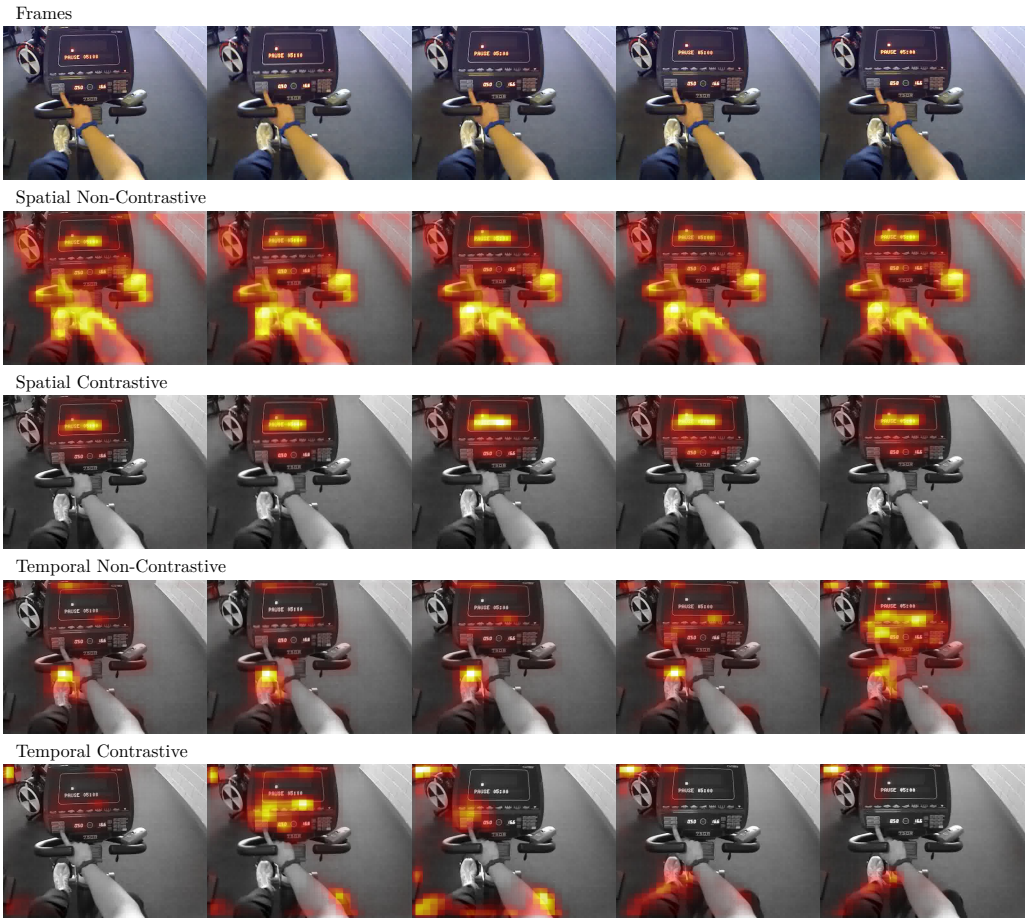


Figure 5.24: BEOID Example: `06_Treadmill1_press_button_4469-4493`

Fig. 5.24 (*press button*, exercise bike): The TNC maps suffer from some jitter, particularly notable between the last two frames where attention shifts from the tip of the shoe to the display on the bike. There is poor localisation of the hand with the regions covering it receiving little attention; this is likely due to the absence of a hand outline in the optical flow as the hand doesn't move during the action. The left shoe receives the majority of attention throughout the first 4 frames since it is the source of the dominant motion in the clip. The movement of the shoe acts as a good discrimantor for actions taking place on the exercise bike. The TC maps suffer from jitter making them difficult to infer features learnt. Constant throughout the TC maps is an area in the top right corner which consistently receives attention; the video exhibits side to side motion during the press-button action as the person peddles. The side to side motion causes the corner of the machine in the top left to come in and out of view resulting in noisy spikes in the corresponding optical flow, and so the network has learnt to use this noise as a proxy indicator for this action.

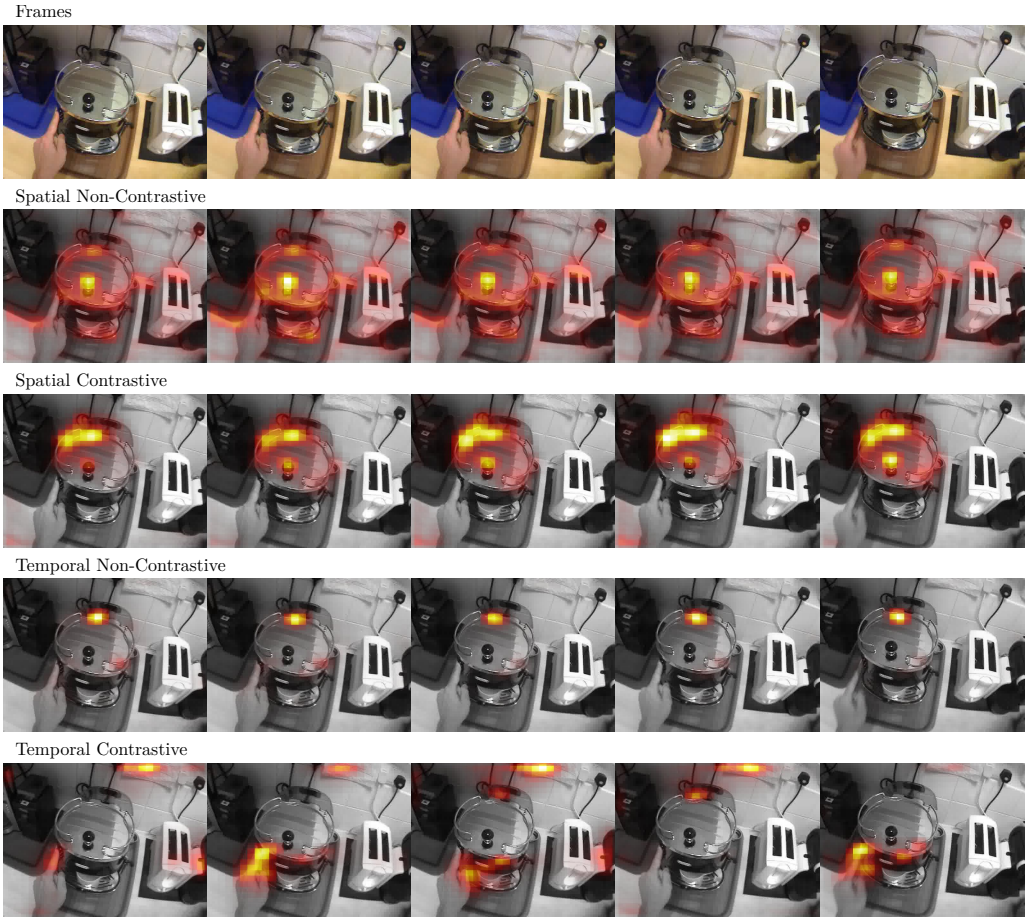


Figure 5.25: BEOID Example: `01_Sink2_press_button_527-561`

Fig. 5.25 (*press button*, coffee machine): The SNC maps distribute attention across the whole frame localising boundaries of objects indicating the network has learnt to recognise the visual appearance of the location. There are strong regions of attention over the handle on the mirrored top of the coffee machine which produces good contrast making a good feature to learn to recognise the coffee machine. There is little attention over the hand suggesting that *environment* has been learnt instead of the appearance of the hand during the action. The SC maps have low jitter further localising attention to the top handle and the back of the mirrored top of the coffee machine. Similarly to the SNC maps, the

SC maps fail to localise the hand. The TNC maps localise almost all the attention to a small region at the back of the mirrored top of the coffee machine, similar to the regions localised by the SC maps. The TC maps suffer from jitter but do localise the button press in three of the five frames; specifically the first, second and fifth.

The *press button* action is presented across three locations: *treadmill*, *exercise bike* and *coffee machine* in fig. 5.23, fig. 5.24, and fig. 5.25 respectively. The SNC maps for the *treadmill* and *exercise bike* locations exhibit significant attention over the hand, given that multiple actions are performed with similar POV in these locations the spatial network has learnt to recognise both the location and the appearance of the hand and forearm during the press button action. In the *coffee machine* location, there is not attention over the hand for either of the spatial attention map sequences indicating the network has learnt the environmental appearance alone, or that the environmental appearance is sufficient and more discriminative than the outstretched hand. The temporal non-contrastive attention maps for the *treadmill* location localise the press button action well, but not in either of the other two locations. The temporal contrastive maps localise the treadmill dashboard, but not so much the dashboard on the exercise bike. Unlike the exercise machines, the temporal contrastive maps for the coffee machine localise the press button in a majority of the frames.

Fig. 5.26 (*turn tap*, sink): All but the TC maps have low jitter. The SNC maps localise the rim of the cup, the washing up liquid bottle logo, the visible tap handle, and the boundaries of the arm and hand. The strong regions of attention surrounding the cup rim suggest the spatial network has learnt to recognise the cup. The washing up bottle logo acts as a strong visual indicator that location is the sink helping refine the number of possible action classes taking place. The environment surrounding the outstretched arm is necessary to infer the action, without it the appearance of the arm is insufficient to distinguish between actions. The SC maps localise the majority of attention to the red tap handle indicating this is a discriminator for the 'turn tap' action. The TNC maps localise attention to the changing reflections of the water in the mug, these cause strong regions in the corresponding optical flow typical of moving water. Some attention is localised to the movement of the arm used to turn the tap in the TNC map. The TC maps have high jitter but do localise the hand and wrist movement in 2 of the 5 frames: the first and fifth frame.

Fig. 5.27 (*stir spoon*, microwave): The TNC and TC maps both have low jitter and are very similar in attention localisation. The attention is primarily distributed to the movement of the spoon bowl in the water in the cup, the TNC maps also localise some hand motion unlike the TC maps.

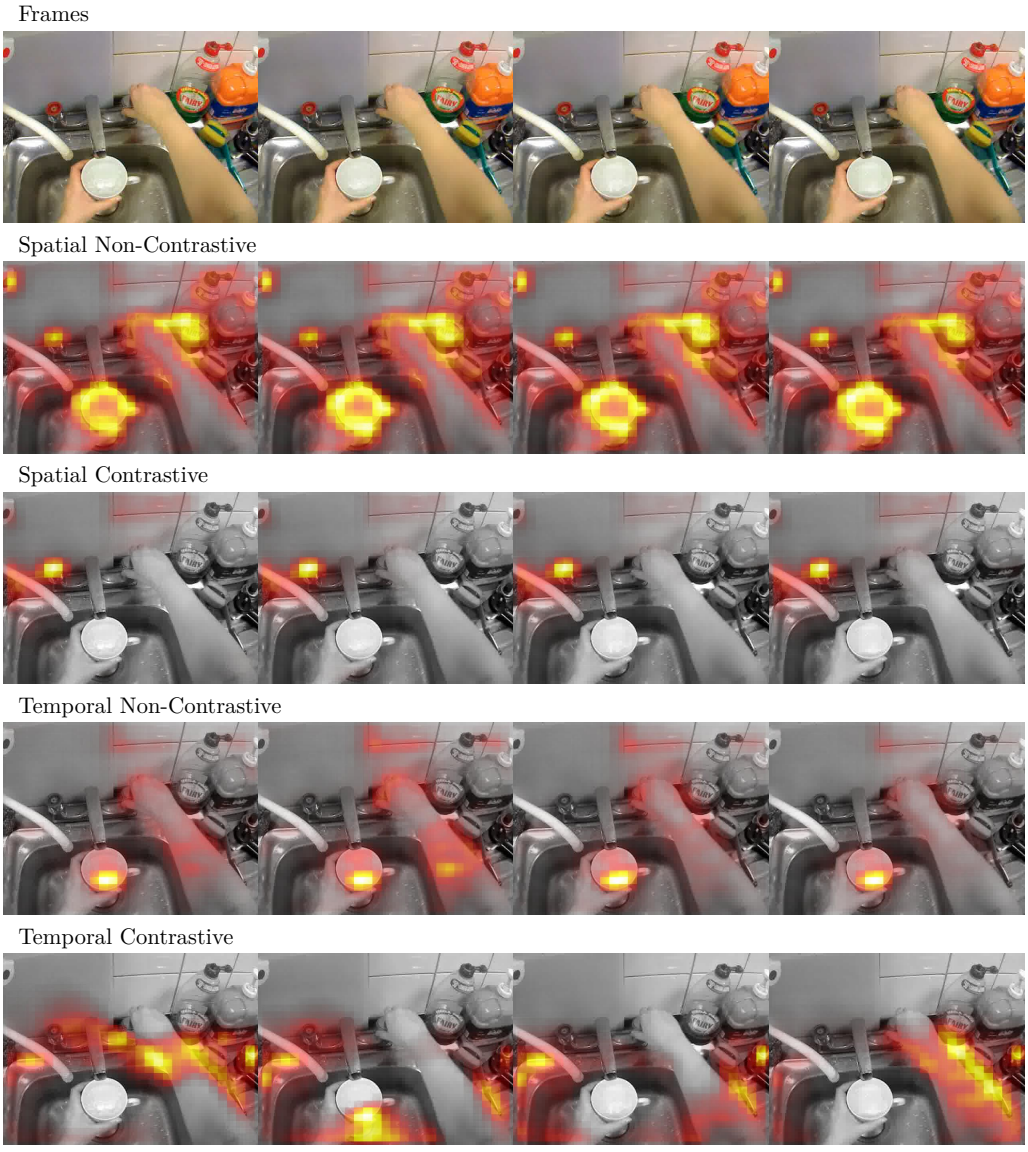


Figure 5.26: BEOID Example: `00_Sink1_turn_tap_694-717`

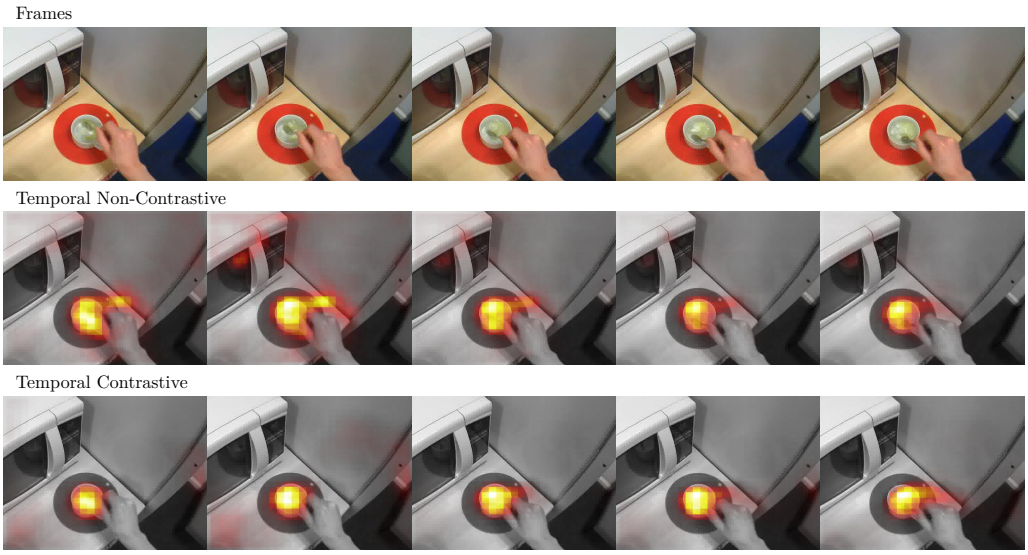


Figure 5.27: BEOID Example: `03_Sink2_stir_spoon_1793-1887`

Summary

In this section we have analysed selected examples representative of the attention map sequences generated for both the UCF101 and BEOID datasets. We have show examples ranging from those with excellent attention map sequences across all EBP methods for both the spatial and temporal streams (e.g. fig. 5.18) to those where the attention maps are difficult to interpret (fig. 5.15). We have seen that non-contrastive EBP usually produces more intelligible attention map sequences compared to contrastive EBP (e.g. figs. 5.13, 5.24, 5.26) although there are some cases where contrastive produces more interpretable sequences at the cost of increased jitter (fig. 5.25). Attention maps created with non-contrastive EBP are easily interpretable for they show the regions salient to a classification. Contrastive EBP highlights the regions that discriminate between classes, alone they are hard to interpret, but combined with comparing to non-contrastive maps they can provide additional information although the high jitter across frames makes it difficult to determine whether they provide accurate information.

6 Conclusion

In this thesis we analysed features learnt by the 2SCNN architecture trained for action recognition on two datasets: BEOID and UCF101 using filter analysis and EBP.

In sec. 3 we made a comprehensive survey of visualisation methods for CNNs with a focus on attention mapping methods that produce a heat map from a network and its input indicating the regions salient to the activation of a chosen neuron. We chose EBP for the analysis of our networks providing a detailed explanation of the method in sec. 4.

In sec. 5.2 We qualitatively analysed the filters learnt in the first and second layer finding that the two streams share very similar filters in the second layer, and that the filters are in the first layer of the temporal network are mostly homogenous across the temporal dimension of the input.

In sec. 5.3 we proposed a method for generating attention map sequences on a per frame basis with a temporal view equal to that of the network input's temporal length by using a sliding window over the video clip from which we derive input to the network, we discussed the implication of the frame underlay for the attention map demonstrating how different frames affect the appearance of the attention map.

In sec. 5.4 we generated attention map sequences for each clip in the test set for both network streams[1] and qualitatively analysed a selection of the attention map sequences to infer features learnt by the network noting pathologies where present in sec. 5.4.4.

We note that the attention map sequences produced by contrastive EBP tend to suffer from *jitter* where similar frames produce considerably different attention maps, we quantify this using L2-jitter showing a significant difference for the spatial network stream trained on UCF101 in sec. 5.4.2.

In sec. 5.4.3 we performed an analysis of the quality of the attention maps by comparing the attention map peak to centre of gaze acting a proxy variable for the action location. We reported the distribution of distances for each network stream and EBP type, also providing a further breakdown by clip location for BEOID.

Contribution summary:

- Survey of visualisation methods for CNNs organised into hierarchy.
- Visual analysis of the first and second layer filters learnt by each stream.
- Extension of EBP for use on the temporal network streams to generate attention maps on a per frame basis.
- A quantitative assessment of attention map sequences for *jitter* demonstrating the inferiority of contrastive EBP on the spatial network for UCF101.
- A quantitative assessment of attention map quality on the BEOID dataset by comparing the distance between action location (using gaze location as a proxy) and attention map peak location.

[1] made available on YouTube: https://goo.gl/QBYZLJ, https://goo.gl/PazivH

- A qualitative assessment of successful and pathological attention map sequences for both datasets.

7 Future work

We present three directions for future work: A comparison of CNN architectures for action recognition using EBP to generate attention maps; an investigation of activation maximisation techniques applied to the 2SCNN architecture to synthesize videos; and an investigation into the effects of class overlap on the quality of attention maps generated by contrastive EBP.

Architecture comparison by attention mapping, A number of architectures for action recognition have been proposed in addition to the late fusion 2SCNN analysed in this thesis, we propose a survey of a set of architectures trained on the same dataset using EBP to generate attention maps. By comparing the attention maps for the same clips across network architectures we can identify those architectures that produce pathological attention maps to gain insight into how the differences in architecture affects the learnt features. In the following two paragraphs we outline two sets of architectures to which this analysis could be applied.

In [10], Feichtenhofer *et al.* extend the 2SCNN architecture, introduced by Simonyan *et al.* in [40] by observing that the previous architecture is incapable of matching appearance in one sub-region to motion in another sub-region since each stream is separate. To remedy this they introduce a modified 2SCNN architecture in which the two streams are combined after the last convolutional layer forming a single spatio-temporal stream from the first fully connected layer onwards. The authors find that keeping the spatial stream in addition to the spatio-temporal stream and combining their respective predictions further increases performance over predictions from the spatio-temporal stream alone.

Karpathy *et al.* investigate architectures for video classification operating on a sequence of video frames (unlike the 2SCNN which operates on both frames and optical flow) in [18]. Four different architectures derived from the same basic architecture are investigated to determine optimal stages of fusing temporal and spatial information. Each architecture has a different connectivity to the video sequence stack, from using a single frame as input to a dense sub-sequence of frames, these architectures are presented in fig. 7.1.

Activation maximisation for temporal networks, To our knowledge, very few researchers have used activation maximisation to visualise features learnt by temporal networks. Wang *et al.* use DeepDraw[1] (an implementation of activation maximisation) to visualise temporal segment networks[47], however they only generate single frames in optical flow space. We propose research into generation of video sequences using activation maximisation by generating frames both in the spatial and temporal domains combining the two to produce a video sequence.

Contrastive attention mapping The application of contrastive EBP to 2SCNN yielded attention map sequences with high jitter. The maps in the generated sequence tend to highlight a few select regions flipping the attention distribution between those regions. We believe this is due to overlap in classes; we suggest an investigation into using prior

[1] https://github.com/auduno/deepdraw

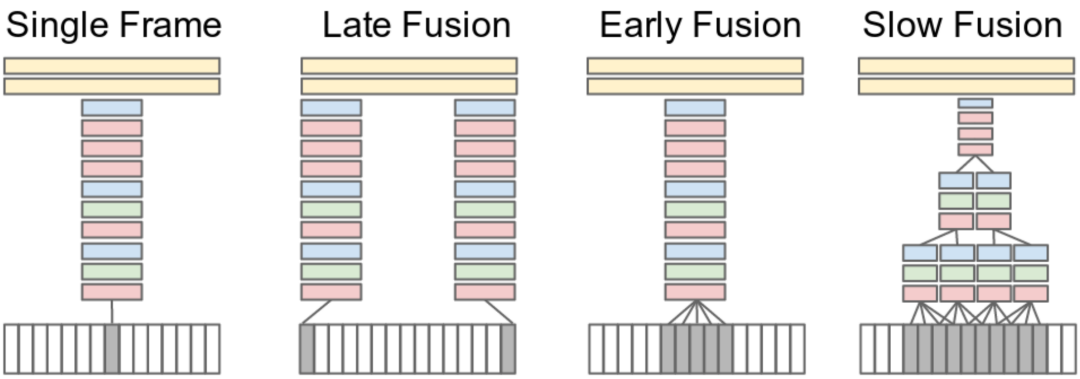


Figure 7.1: CNN Architectures evaluated in [18], layer colours indicate function: convolutional, normalization, pooling, fully connected. The bottom white boxes indicate a series of frames that are used as input to the CNN

distributions over the classification layer in which classes with overlap are set to non-zero probabilities to help determine whether this is the case.

8 Appendices

8.1 Network details

In this section we discuss the details of the networks used for generating attention maps for our evaluation of EBP on temporal networks.

The pretrained network models trained on UCF101 and BEOID were obtained from CUHK[48] and UoB[28] respectively. The UCF101 trained network was trained using ImageNet weight initialisation for both network streams. The BEOID trained network was trained initialising weights from the UCF101 trained model to reduce overfitting as BEOID is a small dataset. The accuracy of the network on the two datasets is detailed in tbl. 5.1.

We build on top of the EBP example code provided by Zhang *et al.* available on GitHub[1], making our work available on GitHub[2]. We use Caffe[17], a neural network framework written in C++ with Python bindings. Excitation backprop is defined as an additional backprop mode in Caffe.

8.2 Jitter extrema

The following tables (tbl. 8.1 and tbl. 8.2) summarise the clips leading to extreme jitter values across each network stream and EBP type. Corresponding videos showing attention maps for can be found on YouTube (link provided at start of sec. 5.4)

Table 8.1: Jitter extrema (UCF101)

Network	EBP Type	Extrema	Clip	Jitter
Spatial	Non-contrastive	Min	`v_PlayingTabla_g07_c01`	2.9
		Max	`v_Mixing_g05_c04`	52.7
	Contrastive	Min	`v_PlayingGuitar_g05_c01`	31.0
		Max	`v_Swing_g06_c07`	126.8
Temporal	Non-contrastive	Min	`v_Hammering_g07_c05`	9.4
		Max	`v_Knitting_g07_c05`	38.6
	Contrastive	Min	`v_RopeClimbing_g05_c07`	8.1
		Max	`v_Haircut_g06_c01`	44.6

[1] https://github.com/jimmie33/Caffe-ExcitationBP
[2] https://github.com/willprice/two-stream-action-cnn-analysis

Table 8.2: Jitter extrema (BEOID)

Network	EBP Type	Extrema	Clip	Jitter
Spatial	Non-contrastive	Min	03_Sink2_stir_spoon_1793-1887	9.7
		Max	02_Sink2_pick-up_jar_1003-1027	52.7
	Contrastive	Min	06_Treadmill1_press_button_4469-4493	5.7
		Max	05_Row1_pull_rowing-machine_2751-2784	46.4
Temporal	Non-contrastive	Min	01_Sink2_press_button_527-561	4.2
		Max	01_Sink1_turn_tap_406-441	39.7
	Contrastive	Min	04_Sink1_press_button_800-835	3.6
		Max	02_Sink1_scoop_spoon_1294-1332	53.8

9 Glossary

ANN Artificial Neural Network
CNN Convolutional Neural Network
DNN Deep artificial neural network (one with multiple hidden layers)
EBP Excitation Backpropagation
2SCNN Two stream CNN, a CNN architecture developed for action recognition composed of two separate network towers: the spatial stream, recognising actions based on appearance; and the temporal stream, recognising actions based on a stack of optical flow.
Filter A filter/kernel in the context of convolutional
Kernel See *Filter*
Clip A short sequence of a video in which an action is performed
Optical flow 2D image of the apparent motion between two frames
Top down attention Attention driven by top down factors like task information
Bottom up attention Attention based on the salience of regions of the input image.
Attention Map A heat map over an image denoting the regions contributing to excitation of a chosen neuron.
Activation maximisation A class of visualisation techniques synthesising an input that maximally activates a chosen neuron
Feature map inversion A class of visualisation techniques similar to *activation maximisation* which synthesize an input to a network given a feature map at a specific layer. The input is synthesised such that it produces the same feature map.
Caricaturing A class of visualisation techniques taking an image and modifying it to further boost the activation of highly activated neurons. Similar to *layer code inversion* and *activation maximisation.*
Layerwise relevance propagation An *attention mapping* method producing discriminative attention maps.
Excitation backpropagation (EBP) An *attention mapping* method capable of computing salient and discriminative attention maps
SC Contrastive attention maps from the spatial stream of the 2SCNN.
SNC Non-contrastive attention maps from the spatial stream of the 2SCNN.
TC Contrastive attention maps from the temporal stream of the 2SCNN.
TNC Non-contrastive attention maps from the temporal stream of the 2SCNN.

10 Notation

A full listing of all notation used.

η Learning rate (e.g. see algorithm 1)

$a_j^{(l)}$ Neuron in layer l (0 indexed) at position j (0 indexed)

$\tilde{a}_j^{(l)}$ The input to neuron $a_j^{(l)}$.

$\hat{a}_j^{(l)}$ The output of neuron $a_j^{(l)}$

$w_{j,k}^{(l)}$ The weight connecting neuron $a_j^{(l)}$ to neuron $a_k^{(l+1)}$

$\mathcal{C}_j^{(l)}$ The child neurons (those in layer $l-1$) of the neuron in layer l with index j $\mathcal{C}_j^{(l)} = \{a_k^{(l-1)} | w_{k,j}^{(l-1)} \neq 0\}$

$\mathcal{P}_j^{(l)}$ The parent neurons (those in layer $l+1$) of the neuron in layer l with index j $\mathcal{P}_j^{(l)} = \{a_k^{(l+1)} | w_{j,k}^{(l)} \neq 0\}$

$P\left(a_j^{(l)} | a_k^{(l+1)}\right)$ The *conditional winning probability* of $a_j^{(l)}$ given that $a_k^{(l+1)}$ is winner neuron (see EBP).

$Z_j^{(l)}$ The scaling factor used in calculating the conditional probabilities in EBP ensuring that the probabilities sum to one.

$P\left(a_j^{(l)}\right)$ The *marginal winning probability* of $a_j^{(l)}$ (see EBP)

A The attention map generated as a result of EBP, it is equivalent to the marginal winning probability distribution over the stopping layer.

$[a \,..\, b]$ The integer range from a to b (inclusive): $\forall x \in [a \,..\, b] : x \geq a \wedge x \leq b \wedge x \in \mathbb{Z}$

Bibliography

[1] Moez Baccouche et al. "Sequential Deep Learning for Human Action Recognition". In: *2nd International Workshop on Human Behavior Understanding (HBU)*. Ed. by B. Lepri A.A. Salah. Amsterdam, Netherlands: Springer, Nov. 2011, pp. 29–39.

[2] Sebastian Bach et al. "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation". In: *PLoS ONE* 10.7 (July 10, 2015). ISSN: 1932-6203.

[3] *Bristol Egocentric Object Interactions Dataset*. URL: http://data.bris.ac.uk/data/dataset/o4hx7jnmfqt01lyzf2n4rchg6 (visited on 03/16/2017).

[4] Charles E. Connor, Howard E. Egeth, and Steven Yantis. "Visual Attention: Bottom-Up Versus Top-Down". In: *Current Biology* 14.19 (Oct. 5, 2004), R850–R852. ISSN: 0960-9822.

[5] Dima Damen, Teesid Leelasawassuk, and Walterio Mayol-Cuevas. "You-Do, I-Learn". In: *Comput. Vis. Image Underst.* 149 (C Aug. 2016), pp. 98–112. ISSN: 1077-3142.

[6] Alexey Dosovitskiy and Thomas Brox. "Inverting Visual Representations with Convolutional Networks". In: (June 8, 2015).

[7] Stuart E. Dreyfus. "Artificial Neural Networks, Back Propagation, and the Kelley-Bryson Gradient Procedure". In: *Journal of Guidance, Control, and Dynamics* 13.5 (1990), pp. 926–928. ISSN: 0731-5090.

[8] Dumitru Erhan et al. *Visualizing Higher-Layer Features of a Deep Network*. Jan. 2009.

[9] Mark Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *Int. J. Comput. Vision* 88.2 (June 2010), pp. 303–338. ISSN: 0920-5691.

[10] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. "Convolutional Two-Stream Network Fusion for Video Action Recognition". In: (Apr. 22, 2016).

[11] Kunihiko Fukushima. "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". In: *Biological Cybernetics* 36.4 (Apr. 1, 1980), pp. 193–202. ISSN: 0340-1200, 1432-0770.

[12] Georgia Gkioxari, Ross Girshick, and Jitendra Malik. "Contextual Action Recognition with R*CNN". In: (May 5, 2015).

[13] Geoffrey E. Hinton. "Deep Belief Networks". In: *Scholarpedia* 4.5 (May 31, 2009), p. 5947. ISSN: 1941-6016.

[14] D. H. Hubel and T. N. Wiesel. "Receptive Fields of Single Neurones in the Cat's Striate Cortex". In: *The Journal of Physiology* 148.3 (Oct. 1959), pp. 574–591. ISSN: 0022-3751.

[15] *Inceptionism: Going Deeper into Neural Networks*. July 17, 2015. URL: https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html (visited on 03/14/2017).

[16] Shuiwang Ji et al. "3D Convolutional Neural Networks for Human Action Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.1 (Jan. 2013), pp. 221–231. ISSN: 0162-8828, 2160-9292.

[17] Yangqing Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: (June 20, 2014).

[18] Andrej Karpathy et al. “Large-Scale Video Classification with Convolutional Neural Networks”. In: IEEE, June 2014, pp. 1725–1732. ISBN: 978-1-4799-5118-5.
[19] Henry J. Kelley. “Gradient Theory of Optimal Flight Paths”. In: *ARS Journal* 30.10 (1960), pp. 947–954.
[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “Imagenet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. 2012, pp. 1097–1105.
[21] H. Kuehne et al. “HMDB: A Large Video Database for Human Motion Recognition”. In: *2011 International Conference on Computer Vision*. 2011 International Conference on Computer Vision. Nov. 2011, pp. 2556–2563.
[22] Michael F. Land and Mary Hayhoe. “In What Ways Do Eye Movements Contribute to Everyday Activities?” In: *Vision Research* 41 (25–26 Nov. 2001), pp. 3559–3565. ISSN: 0042-6989.
[23] Laurens van der Maaten and Geoffrey Hinton. “Visualizing Data Using T-SNE”. In: *Journal of Machine Learning Research* 9 (Nov 2008), pp. 2579–2605. ISSN: ISSN 1533-7928.
[24] Aravindh Mahendran and Andrea Vedaldi. “Understanding Deep Image Representations by Inverting Them”. In: (Nov. 26, 2014).
[25] Aravindh Mahendran and Andrea Vedaldi. “Visualizing Deep Convolutional Neural Networks Using Natural Pre-Images”. In: *International Journal of Computer Vision* 120.3 (Dec. 2016), pp. 233–255. ISSN: 0920-5691, 1573-1405.
[26] Warren S. McCulloch and Walter Pitts. “A Logical Calculus of the Ideas Immanent in Nervous Activity”. In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1, 1943), pp. 115–133. ISSN: 0007-4985, 1522-9602.
[27] M. Minsky and S. Papert. *Perceptrons*. Cambridge, MA: MIT Press, 1969.
[28] Davide Moltisanti et al. “Trespassing the Boundaries: Labeling Temporal Bounds for Object Interactions in Egocentric Video”. In: (Mar. 27, 2017).
[29] Anh Nguyen, Jason Yosinski, and Jeff Clune. “Deep Neural Networks Are Easily Fooled: High Confidence Predictions for Unrecognizable Images”. In: (Dec. 5, 2014).
[30] Anh Nguyen, Jason Yosinski, and Jeff Clune. “Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks”. In: (Feb. 11, 2016).
[31] Anh Nguyen et al. “Synthesizing the Preferred Inputs for Neurons in Neural Networks via Deep Generator Networks”. In: (May 30, 2016).
[32] Suneth Ranasinghe, Fadi Al Machot, and Heinrich C Mayr. “A Review on Applications of Activity Recognition Systems with Regard to Performance and Evaluation”. In: *International Journal of Distributed Sensor Networks* 12.8 (Aug. 1, 2016), p. 1550147716665520. ISSN: 1550-1477.
[33] T. Rose et al. “The TRECVid 2008 Event Detection Evaluation”. In: *2009 Workshop on Applications of Computer Vision (WACV)*. 2009 Workshop on Applications of Computer Vision (WACV). Dec. 2009, pp. 1–8.
[34] Frank F. Rosenblatt. “The Perceptron: A Perceiving and Recognising Automaton (Project PARA)”. In: *Cornell Aeronautical Laboratory* (1957).
[35] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: (Sept. 1, 2014).
[36] Wojciech Samek et al. “Evaluating the Visualization of What a Deep Neural Network Has Learned”. In: (Sept. 21, 2015).
[37] Juergen Schmidhuber. “Deep Learning in Neural Networks: An Overview”. In: *Neural Networks* 61 (Jan. 2015), pp. 85–117. ISSN: 08936080.

[38] C. Schuldt, I. Laptev, and B. Caputo. “Recognizing Human Actions: A Local SVM Approach”. In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004. Vol. 3. Aug. 2004, 32–36 Vol.3.

[39] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”. In: (Dec. 20, 2013).

[40] Karen Simonyan and Andrew Zisserman. “Two-Stream Convolutional Networks for Action Recognition in Videos”. In: (June 9, 2014).

[41] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: (Sept. 4, 2014).

[42] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. “UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild”. In: (Dec. 3, 2012).

[43] Christian Szegedy et al. “Going Deeper with Convolutions”. In: (Sept. 16, 2014).

[44] *TRECVID Data.* URL: http://trecvid.nist.gov/trecvid.data.html (visited on 03/14/2017).

[45] John K. Tsotsos et al. “Modeling Visual Attention via Selective Tuning”. In: *Artif. Intell.* 78 (1-2 Oct. 1995), pp. 507–545. ISSN: 0004-3702.

[46] Pascal Vincent et al. “Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion”. In: *The Journal of Machine Learning Research* 11 (Mar. 1, 2010), pp. 3371–3408. ISSN: 1532-4435.

[47] Limin Wang et al. “Temporal Segment Networks: Towards Good Practices for Deep Action Recognition”. In: (Aug. 2, 2016).

[48] Limin Wang et al. “Towards Good Practices for Very Deep Two-Stream ConvNets”. In: (July 8, 2015).

[49] Jason Yosinski et al. “Understanding Neural Networks Through Deep Visualization”. In: (June 22, 2015).

[50] Wei Yu et al. “Visualizing and Comparing Convolutional Neural Networks”. In: (Dec. 20, 2014).

[51] C. Zach, T. Pock, and H. Bischof. “A Duality Based Approach for Realtime TV-L1 Optical Flow”. In: *Proceedings of the 29th DAGM Conference on Pattern Recognition.* Berlin, Heidelberg: Springer-Verlag, 2007, pp. 214–223. ISBN: 978-3-540-74933-2.

[52] M. D. Zeiler et al. “Deconvolutional Networks”. In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. June 2010, pp. 2528–2535.

[53] Matthew D. Zeiler and Rob Fergus. “Visualizing and Understanding Convolutional Networks”. In: (Nov. 12, 2013).

[54] Jianming Zhang et al. “Top-down Neural Attention by Excitation Backprop”. In: (Aug. 1, 2016).