DDLSTM: Dual-Domain LSTM for Cross-Dataset Action Recognition

Toby Perrett and Dima Damen
Department of Computer Science, University of Bristol, Bristol, UK
<firstname>.<lastname>@bristol.ac.uk

Abstract

Domain alignment in convolutional networks aims to learn the degree of layer-specific feature alignment beneficial to the joint learning of source and target datasets. While increasingly popular in convolutional networks, there have been no previous attempts to achieve domain alignment in recurrent networks. Similar to spatial features, both source and target domains are likely to exhibit temporal dependencies that can be jointly learnt and aligned.

In this paper we introduce Dual-Domain LSTM (DDLSTM), an architecture that is able to learn temporal dependencies from two domains concurrently. It performs cross-contaminated batch normalisation on both input-to-hidden and hidden-to-hidden weights, and learns the parameters for cross-contamination, for both single-layer and multi-layer LSTM architectures. We evaluate DDLSTM on frame-level action recognition using three datasets, taking a pair at a time, and report an average increase in accuracy of 3.5%. The proposed DDLSTM architecture outperforms standard, fine-tuned, and batch-normalised LSTMs.

1. Introduction

Online action recognition has direct implications on assistive and surveillance applications, enabling action classification as soon as a new frame is observed. It only depends on previously observed frames, with no knowledge from future observations. This contrasts offline action recognition where the whole action is observed before being classified.

One obstacle to deploying online action-recognition systems in the wild is that they require a large amount of training data to achieve high performance, so an open question is how to make best-use of multiple datasets to achieve higher recognition accuracy. Such cross-dataset temporal dependencies can be present even when datasets use different class labels. In this paper, we focus on the related tasks of kitchen activities. In one dataset, a sequence of actions could be labeled as:

‘pick-up knife’ $\rightarrow$ ‘cut onion’ $\rightarrow$ ‘put onion in pan’

whereas a second dataset would have labels such as:

‘take knife’ $\rightarrow$ ‘chop potato’ $\rightarrow$ ‘place on a baking tray’

While the two sets of labels differ, we investigate how a joint recurrent model can be learnt for both datasets and we demonstrate that this joint training outperforms independently-learnt models.

In online recognition, recurrent models are typically used, particularly Long Short-Term Memory recurrent networks (LSTM) [9, 8, 27]. It has recently been shown that LSTMs can benefit from multiple data sources when using CNN features for frame-based action classification [32]. However, LSTMs are not explicitly designed to handle information from multiple domains. We aim to address this limitation by combining recent advances in CNN domain adaptation [2] with batch-normalised LSTM training [4], and introduce the Dual-Domain LSTM (DDLSTM). We show that it is indeed possible for the DDLSTM to learn jointly from two related datasets, and in a way which outperforms the standard LSTM for frame-based online action recognition (whether jointly trained, or pre-trained and fine-tuned). Importantly, our formalism allows learning cross-contamination parameters in a differentiable manner within the backpropation-through-time of the LSTM. In each experiment, we evaluate DDLSTM on pairs of datasets, out of three datasets frequently used for kitchen-based activities: 50 Salads [40], Breakfast [21] and MPII Cooking 2 [36]. We also demonstrate the benefit of joint training with larger datasets of related domains (e.g. EPIC [6]) to leverage missing temporal knowledge.

The rest of this paper is organised as follows. Section 2 gives a summary of related domain adaptation and LSTM literature. Section 3 introduces the proposed DDLSTM. Section 4 gives an overview of the datasets we use for evaluation. Section 5 includes the comparative analysis, where we show an average increase in frame-level recognition accuracy of 3.5%. Finally, the conclusion is in Section 6.

2. Background

Works which learn from multiple domains have traditionally used measures, such as the Maximum Mean Discrepancy (MMD) [13, 28], to determine differences in feature space between the domains, and apply transformations to bring them closer together [1, 33]. More recently, loss functions which take these differences into account have been used, in conjunction with classification loss, in CNNs to merge domains in an end to end fashion [14, 5, 42], enabling predictions to be made on unsupervised domains. Recent advances have taken the approach of introducing additional layers into a network in order to align deep and shallow feature spaces. Of particular interest to us is the work of Carlucci et al. [2]. Their Automatic Domain Alignment Layers are based around expanding batch normalisation (introduced by Ioffe and Szegedy [18] to improve model accuracy and reduce the number of training iterations) to handle inputs from source and target domains. This is achieved by calculating separate batch statistics from source and target samples within a batch, but allowing for some cross-contamination between domains. These layers are inserted into classification networks (after the fully connected layers in AlexNet [20] and replacing standard batch normalisation layers in InceptionBN), equipped with a softmax classification loss for source samples and an entropy loss for unlabelled target samples. This approach is related to, but fundamentally different from cross-network stitching for multi-tasks [29]. In [29], two networks which perform different tasks (e.g. detection vs captioning) on *the same input* are trained. However in [2], one task (e.g. classification) is performed on *two sets input features* which are not necessarily related.

The works covered so far are all designed for classification using convolutional networks trained on single images. For example, a commonly used benchmark is the Office+Caltech dataset [11] where adaptation between images taken with a DSLR in an office and product images taken from Amazon is attempted. The most common technique to utilise multiple datasets in video-level classification is to train on one and fine tune on another [3]. Attempts have also been made to use semantic similarity between labels [44], and to treat different camera angles of the same content as different domains [31].

We are specifically interested in online action recognition, where single-dataset methods tend to use CNN features with LSTMs to make frame-level predictions [30, 43].Incorporating a domain adaptation component into recurrent neural networks (RNNs) has the potential to provide a number of benefits. Applying domain adaptation within an RNN will allow the direct use of different features which are well suited to their respective domains, whilst also enabling the learning of related temporal information from multiple domains at the same time. Additionally, when RNNs are used with raw sensor output, the only place where domain adaptation can occur is within the RNN.

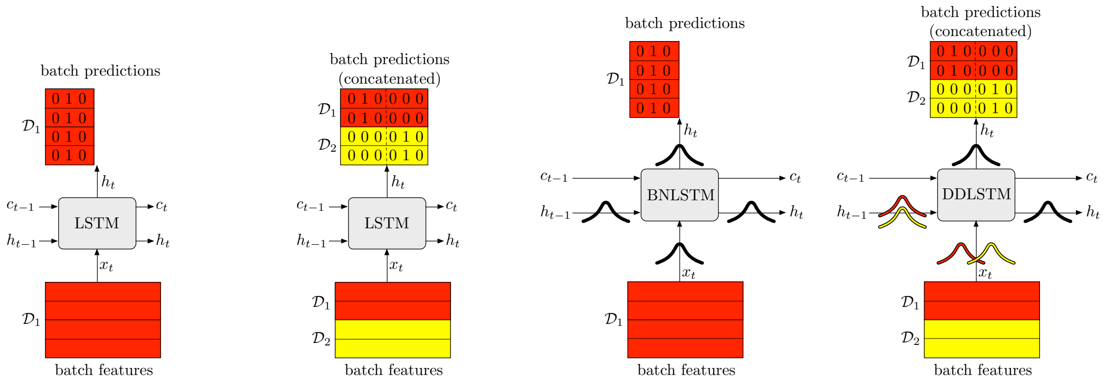
LSTMs are an obvious candidate for modification to become multi-domain aware, as their ability to remember information for a long period of time makes them particularly well suited to applications such as frame-based action recognition (which we focus on here), and language modelling, amongst others. Greff et al. [12] evaluated 8 different LSTM cell types, and found there was very little difference between them. A number of works have attempted to incorporate batch normalisation into an LSTM framework [24, 4]. In [4], Recurrent Batch Normalisation consists of normalisation of two sets of weights; Input-to-hidden weight normalisation can be thought of as standard batch normalisation, and this is used in conjunction with hidden-to-hidden normalisation. They found that performing a separate normalisation for each timestep achieved better performance due to initial activations being dissimilar to values which are converged to after multiple timesteps. We revisit the batch-normalised LSTM (BNLSTM) in Section 3.

3. Dual-Domain LSTM

In this section, we propose the DDLSTM, capable of operating on arbitrary sequential data, which we evaluate in the context of online action recognition. It is able to jointly learn temporal dependencies from two domains: both independent temporal dependencies per domain, as well as common cross-domain temporal dependencies.

Fig. 1a shows a standard LSTM trained on one dataset. Fig. 1b shows a standard LSTM trained jointly on two datasets (i.e. each batch contains examples from both). This generalises to cases where the labels in the two datasets do not necessarily match. Should the labels match, a single shared output vector can be used. However, when each domain has its own set of labels, predictions can be defined as concatenated label vectors, with the first part of the output vector corresponding to predictions for the first dataset's labels, and the second part corresponding to predictions for the second dataset's labels. While this architecture learns the mapping from both input domains to shared or distinct output labels, the model is likely to learn each domain independently, as no effort to align the input from both domains is incorporated in the architecture.

Before introducing the DDLSTM for aligning and jointly learning from two domains, we revisit the BNLSTM for single domain batch normalisation from [4]. Fig. 1c shows the BNLSTM trained on a single dataset. We choose to base the DDLSTM on the BNLSTM architecture for two main reasons. First, the BNLSTM demonstrated superior results over the standard LSTM in applications such as language modelling and simple sequential MNIST [4]. Second, and perhaps more importantly, it incorporates batch normalisation, which makes it suitable for adaptation to work with the domain-mixing aspect (via batch normalisation) of automatic domain alignment layers. Its formulation in [4] is



(a) Standard LSTM trained on one dataset.

(b) LSTM trained jointly on two datasets, with concatenated labels.

(c) BNLSTM trained on one dataset. BN on feature input and hidden-to-hidden input and output.

(d) DDLSTM with concatenated labels. Dual-domain BN on feature and hidden-to-hidden inputs.

Figure 1: Proposed DDLSTM (d) in comparison to other LSTM architectures and training procedures: (a) single domain LSTM, (b) jointly trained LSTM, (c) single domain batch-normalised LSTM.

given as:

$$\begin{pmatrix} \tilde{\mathbf{f}}_t \\ \tilde{\mathbf{i}}_t \\ \tilde{\mathbf{o}}_t \\ \tilde{\mathbf{g}}_t \end{pmatrix} = \begin{array}{l} \mathrm{BN}\left(\mathbf{W}_h\mathbf{h}_{t-1};\gamma_h,\beta_h\right) \\ +\mathrm{BN}\left(\mathbf{W}_x\mathbf{x}_t;\gamma_x,\beta_x\right)+\mathbf{b} \end{array} \tag{1}$$

$$\mathbf{c}_t = \sigma\left(\tilde{\mathbf{f}}_t\right)\odot\mathbf{c}_{t-1}+\sigma\left(\tilde{\mathbf{i}}_t\right)\odot\tanh\left(\tilde{\mathbf{g}}_t\right) \tag{2}$$

$$\mathbf{h}_t = \sigma\left(\tilde{\mathbf{o}}_t\right)\odot\tanh\left(\mathrm{BN}\left(\mathbf{c}_t;\gamma_c,\beta_c\right)\right) \tag{3}$$

Equation 1 contains the forget gate layer ($\tilde{\mathbf{f}}$), input gate layer ($\tilde{\mathbf{i}}$), output gate layer ($\tilde{\mathbf{o}}$) and layer used to generate candidates to change the cell state later on ($\tilde{\mathbf{g}}$). Here, the normalisation of the $\mathbf{W}_x\mathbf{x}_t$ term can be thought of as input-to-hidden normalisation (as it operates on the input at the current timestep, $\mathbf{x}_t$). The normalisation of the $\mathbf{W}_h\mathbf{h}_{t-1}$ term can be thought of as hidden-to-hidden normalisation, as it operates on the output of the cell at the previous timestep ($\mathbf{h}_{t-1}$). Equation 2 gives the new cell state $\mathbf{c}_t$. Note how it is not normalised, which allows the gradient to flow through timesteps. Equation 3 gives the cell output, where $\mathbf{c_t}$ is normalised to match the $\tilde{\mathbf{o}}$ term. The batch normalisation function [4] is:

$$\mathrm{BN}\left(\mathbf{h};\gamma,\beta\right) = \beta+\gamma\odot\frac{\mathbf{h}-\widehat{\mathbb{E}}[\mathbf{h}]}{\sqrt{\widehat{\mathrm{Var}}[\mathbf{h}]+\epsilon}} \tag{4}$$

where β and γ are the offset and scale, fixed at 0 and 0.1 in practice. The BNLSTM above assumes the observations come from a single domain (Fig. 1c). We next propose to expand this to work on samples from two domains, $\mathcal{D}_1$, $\mathcal{D}_2$.

Fig. 1d introduces DDLSTM, proposed in this paper, which is designed to jointly learn temporal dependencies in two datasets, utilising the power of LSTMs in learning both short- and long-term dependencies. We do this by first using concatenated label vectors. If $\mathcal{D}_1$ has L_1 labels and $\mathcal{D}_2$ has L_2 labels, then the one-hot label vector provided to the LSTM will be $L_1 + L_2$ dimensional. Labels corresponding to $\mathcal{D}_1$ occupy entries 0 to $L_1 - 1$, and labels corresponding to $\mathcal{D}_2$ occupy entries L_1 to $L_1 + L_2 - 1$. Another, more crucial, modification is to construct each batch with samples from $\mathcal{D}_1$ *and* $\mathcal{D}_2$, so the BNLSTM can be jointly trained on both. The first n_1 samples in the batch are from $\mathcal{D}_1$, while the rest ($n_2 = N - n_1$) are from $\mathcal{D}_2$.

The standard BNLSTM is not suitable for dual domains, as the two domains are likely to have different means and variances. In order to address this issue, a separate batch normalisation can be performed for samples from each domain. This would be sufficient but would, yet again, ignore any shared (cross-domain) temporal dependencies. We aim to learn cross-contamination between domains, when calculating batch statistics, as follows.

For each domain $\mathcal{D}_i$, we aim to learn a corresponding cross-contamination factor α_i which is used to determine the contributions of samples from the other domain to be included in the mean and variance calculations. Each α_i is constrained such that $n_i \leq \alpha_i N \leq N$. A higher α_i indicates that more cross-contamination occurs and vice versa. Note that this cross contamination is required for accurate variance calculation – if only means were required, then a weighted average of means for $\mathcal{D}_1$ and $\mathcal{D}_2$ could be used. The contribution function, τ_i, then determines the contribution of the j'th sample in the batch for each domain, for a given parameter α_i. Each domain has its own contribution function (remember that samples from $\mathcal{D}_1$ appear first in each batch), defined as:

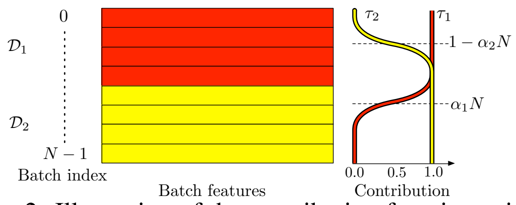


Figure 2: Illustration of the contribution functions given in Equations 5 (red) and 6 (yellow). The contribution of samples from both $\mathcal{D}_1$ and $\mathcal{D}_2$ to $\mathcal{D}_1$'s batch statistics is given by τ_1, which is governed by the variable α_1. Similarly, α_2 governs τ_2, which gives the contribution of samples from both datasets to the batch statics of $\mathcal{D}_2$.

$$\tau_1(\alpha_1, j) = \frac{1 - \tanh(j - \alpha_1 N)}{2} \tag{5}$$

$$\tau_2(\alpha_2, j) = \frac{1 + \tanh(j - \alpha_2 N)}{2} \tag{6}$$

An illustration of this process is given in Fig 2.

This can be used to redefine the batch normalisation function which learns from one domain (Equation 4) with a dual-domain batch normalisation function, DDBN:

$$\text{DDBN}(\mathbf{h}; \gamma, \beta, \alpha_1, \alpha_2) = \beta + \gamma \odot \frac{\mathbf{h} - \widehat{\text{DD}\mathbb{E}}[\mathbf{h}, \alpha_1, \alpha_2]}{\sqrt{\widehat{\text{DDVar}}[\mathbf{h}, \alpha_1, \alpha_2] + \epsilon}} \tag{7}$$

Instead of using standard expectation and variance calculations, DDBN relies on the contribution functions given in Equations 5 and 6 to give the expectation and variance for a weight w at a specific timestep from each $\mathcal{D}_i$ as:

$$\text{DD}\mathbb{E}_i(w) = \frac{\sum_{j=1}^{N} w^j \tau_i(\alpha_i, j)}{\sum_{j=1}^{N} \tau_i(\alpha_i, j)} \tag{8}$$

$$\text{DDVar}_i(w) = \frac{\sum_{j=1}^{N} \left(w^j - \mathbb{E}_i(w)\right)^2 \tau_i(\alpha_i, j)}{\sum_{j=1}^{N} \tau_i(\alpha_i, j)} \tag{9}$$

where w^j denotes the value of w corresponding to the j'th sample in the batch.

The main advantage of operating on the α values with tanh, rather than just selecting samples as in [2], is that it allows the whole process to be differentiable, and α values can be learned as part of the LSTM backpropogation-through-time process. During training, $\text{DD}\mathbb{E}_i$ and DDVar_i are estimated from the batch, and population wide estimates are updated for both domains as more batches are processed. When testing an unseen sample, a flag is passed indicating which dataset the sample belongs to, and the dataset's population estimates are used for normalisation. Fig. 3 shows this training and testing process.

Note in Fig. 1d, two DDBN functions are applied: input-to-hidden, and hidden-to-hidden. While a different set of

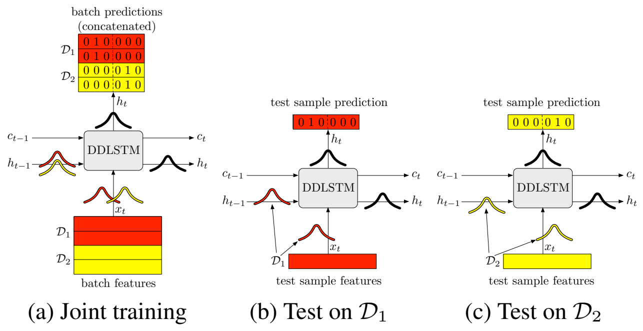


Figure 3: DDLSTM joint training and testing on each dataset in turn. During testing, a flag specifying which dataset the sample comes from is passed to the DDBN layers in order to use the correct batch statistics.

α parameters could be used for each, we found that a single set of parameters gives better performance, presumably because there is similar dataset crossover at the hidden-to-hidden and input-to-hidden stages, and fewer parameters need to be learned. Given these findings, we can define the DDLSTM, highlighting proposed differences in blue, as:

$$\begin{pmatrix} \tilde{\mathbf{f}}_t \\ \tilde{\mathbf{i}}_t \\ \tilde{\mathbf{o}}_t \\ \tilde{\mathbf{g}}_t \end{pmatrix} = \begin{array}{l} \text{DDBN}(\mathbf{W}_h \mathbf{h}_{t-1}; \gamma_h, \beta_h, \alpha_1, \alpha_2) \\ +\text{DDBN}(\mathbf{W}_x \mathbf{x}_t; \gamma_x, \beta_x, \alpha_1, \alpha_2) + \mathbf{b} \end{array} \tag{10}$$

with $\mathbf{c}_t$ and $\mathbf{h}_t$ defined as previously in Equations 2 and 3. DDBN was also trialled instead of the BN function in Equation 3, and found to be be less effective (with regard to both the results and stability during training). We hypothesise that this is because there is little point in effectively performing DDBN on the same data twice, and it would confuse the calculation of the final class probabilities.

Fig. 4 extends the proposed DDLSTM to two layers, and highlights where in the cell different forms of batch normalisation occur. Note that the parameters α are shared between timesteps, along with the rest of the LSTM cell, but the dual-domain batch normalisations are run individually for each timestep. Whilst the number of samples from $\mathcal{D}_1$ (n_1) and $\mathcal{D}_2$ (n_2) in each batch can vary, in our experiments we set $n_1 = n_2 = N/2$. If either n_1 or n_2 are zero (i.e. training data is only drawn from one domain), then the DDLSTM reduces to the BNLSTM [4].

The Multi-layer DDLSTM can benefit from a gradual increase in cross-contamination, from lower to higher layers. This is based on the assumption that more shared information will be present in higher-layer representations than domain-specific lower-layers. This was shown to be the case in CNNs [2] where cross-contamination increases as the network goes from deep to shallow. In Section 5, we test DDLSTM architectures with up to 10 layers, and show 3 layers give the best performance.

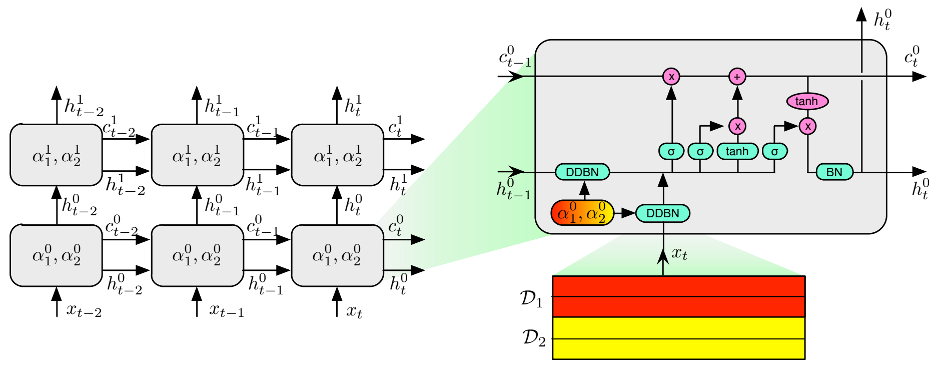


Figure 4: Multi-layer DDLSTM architecture (left), and a DDLSTM cell (right). Note that αs are shared, along with other LSTM parameters, between timesteps of the same level. However, the dual-domain batch normalisation (which uses α values to determine the amount of cross-contamination) is calculated separately for each timestep.

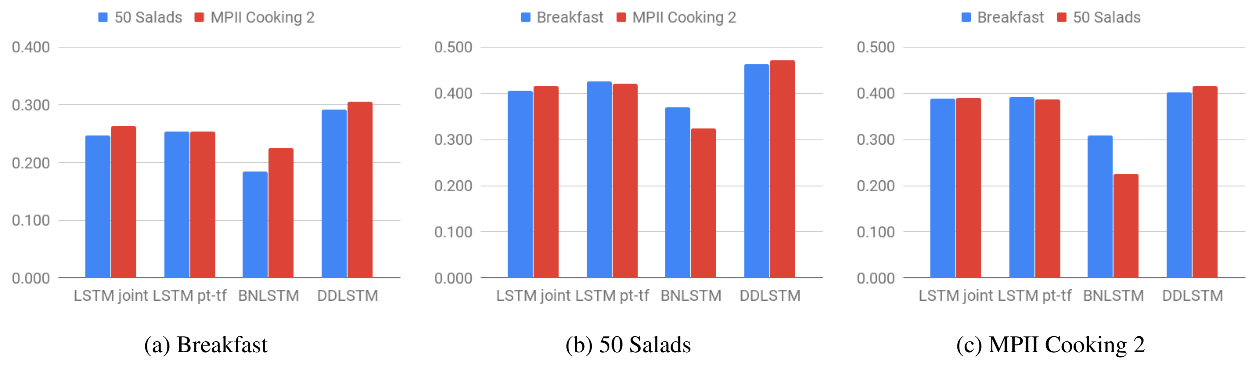


Figure 5: Visualised comparative results for each dataset. Colours indicate the second dataset used in DDLSTM in each case. For example, (a) shows the results for Breakfast trained with 50 Salads (blue) or trained with MPII Cooking (red).

4. Datasets

For this work, we use the three largest datasets of cooking-related activities with framewise action labels, which are all based around activities in the kitchen. These are the Breakfast [21], 50 Salads [40] and MPII Cooking 2 [36] datasets. There is very little label crossover as they are captured in different environments, with different viewpoints, participants and recipes. However, we assume temporal dependences in the common tasks can be leveraged during a shared training process. Some visual examples are given, along with qualitative results of some experiments, in Figure 6. For all datasets in this paper, 4 train/test splits are used, with 75% training and 25% testing. All splits use leave-person-out, i.e. no participant appears in both training and testing sets from the same split.

Note that datasets such as UCF [39], HMDB [23] and Kinetics [3] are not suitable here because they only contain a single action class per video sequence. For online action classification, where each frame is classified as soon as it is seen, multiple actions are required for each video to ensure a robust evaluation. Datasets which would fit this criteria include THUMOS [17] and ActivityNet [15], but they lack the task related-ness of Breakfast, 50 Salads and MPII Cooking, as we show in Section 5.4.

Breakfast [21]: The Breakfast dataset contains 433 sequences performed by 52 participants, containing 3078 actions across 50 classes (including a background class) in 18 different kitchens. All the sequences are one of 10 breakfast routines such as “cooking scrambled eggs” and “making tea”, and no specific recipes are followed. For the experiments in this paper, the lowest level action labels are used. Examples include “pour cereal” and “smear butter”.

50 Salads [40]: The 50 Salads dataset contains 50 videos, by 25 participants. There are 52 of the lowest level action classes (including the background class which we have added), which gives a total of 2967 labelled actions. Example actions include “cut tomato prep”, “cut tomato core”, and “cut tomato post”. These prep- and post- labels are not found in the other two datasets.

MPII Cooking 2 Salads [36]: The MPII Cooking 2 dataset contains 275 sequences from 30 participants in one kitchen. It consists of 14105 actions across 88 classes (in-

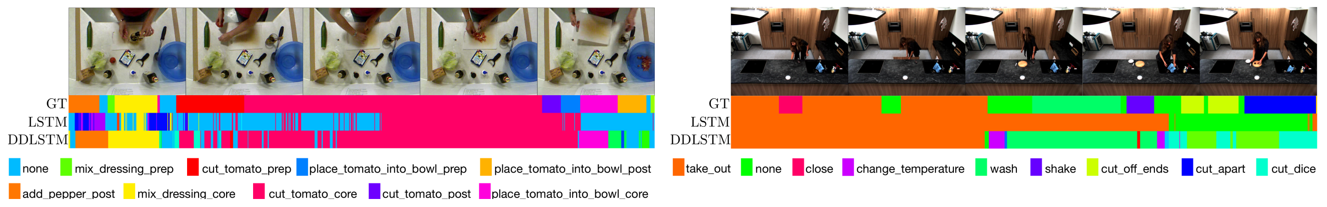


Figure 6: A 1000 frame section from 50 Salads (left) and MPII cooking 2 (right). GT shows the ground truth, LSTM indicates a standard LSTM fine-tuned, and DDLST is jointly trained on (Breakast and 50 Salads left; MPII and 50 Salads right).

D1	D2	Training	LSTM Type	D1 Avg	D2 Avg
50 Salads	MPII Cooking 2	Single	None	41.1	38.3
50 Salads	MPII Cooking 2	Joint	LSTM	41.6	39.0
50 Salads	MPII Cooking 2	Joint	BNLSTM	32.4	22.6
50 Salads	MPII Cooking 2	D1,D2	LSTM	04.4	38.7
50 Salads	MPII Cooking 2	D2,D1	LSTM	43.0	00.0
50 Salads	MPII Cooking 2	Joint	DDLSTM	**47.1**	**41.5**
Breakfast	50 Salads	Single	None	24.5	41.1
Breakfast	50 Salads	Joint	LSTM	24.7	40.5
Breakfast	50 Salads	Joint	BNLSTM	18.4	37.0
Breakfast	50 Salads	D1,D2	LSTM	00.0	42.5
Breakfast	50 Salads	D2,D1	LSTM	25.4	08.5
Breakfast	50 Salads	Joint	DDLSTM	**29.1**	**46.3**
Breakfast	MPII Cooking 2	Single	None	24.5	38.3
Breakfast	MPII Cooking 2	Joint	LSTM	26.3	38.8
Breakfast	MPII Cooking 2	Joint	BNLSTM	22.5	30.9
Breakfast	MPII Cooking 2	D1,D2	LSTM	00.0	39.1
Breakfast	MPII Cooking 2	D2,D1	LSTM	25.3	01.0
Breakfast	MPII Cooking 2	Joint	DDLSTM	**30.5**	**40.1**

Table 1: Average results, over splits, on pairs of the three datasets. A comparison of the Dual-Domain LSTM is compared to BNLSTM and standard LSTM using different training approaches (joint training, pre-training on D1 and fine-tuning on D2 and vice versa). None: frame-level classification without any temporal modelling.

cluding the background class which we have added). Example actions include “shake”, “spread”, and “apply plaster”.

5. Experiments

In this section we detail frame-based feature extraction, and provide comparative analysis against other LSTM architectures. In all experiments, frame-based classification accuracy is reported. **Implementation Details:** For each LSTM type, 128 hidden units are used per cell. We use a batch size of 128 with a 50/50 split between datasets, so each batch contains 64 sequences from each of the two datasets being evaluated. A learning rate of 0.01 is used for 50,000 iterations, and all α values are initialised to 0.75. Other values between 0.5 and 1 were trialled, but made little difference.

5.1. Online frame-based Results

We are particularly interested in online action recognition, that is the ability to recognise actions given current and past observations, without an insight into future frames. The input to each of the tested LSTM architectures are frame-level features extracted from a CNN. For each split (from each dataset), Inception V2 [41] (initialised with HMDB51 weights [23]) is trained using the training split to classify frames individually. This model is then used to extract features for test images. The activations from the last layer of the network (i.e. the logits) are extracted to use as features. Baselines for single datasets are given in Table 1.

It has been shown that directly predicting the next action boundary provides better performance on frame-based classification problems [26]. Following the method for prediction in this work, we first train an LSTM architecture (1 layer deep, history size 200 frames), where the loss for a sequence is the KL divergence between Gaussians at the next action boundary and the current boundary prediction for every frame. This boundary prediction and the original features are then fed as input to the various LSTM architectures (3 layers deep with a history of 200 frames), trained using a softmax loss. Concatenated label vectors, as illustrated earlier in Fig. 1, are used for all experiments. We evaluate frame-based action recognition on the three datasets introduced in Section 4.

The following LSTM architectures and training protocols are compared:

- LSTM jointly trained on two datasets.
- LSTM pre-trained on one and fine-tuned on the other.
- BNLSTM [4] jointly trained on two datasets.
- DDLSTM jointly trained on two datasets.

Table 1 gives the results for these experiments, averaged over all four splits. A corresponding visualisation for these averages can be seen in Fig. 5. It shows that the DDLSTM outperforms the jointly trained and fine-tuned LSTMs as well as the jointly trained BNLSTM across all pairs of datasets. Compared to the next best performing method for each dataset pair, where the second best is a different method for each case, there are increases of 5.1% and 2.5% for 50 Salads and MPII Cooking 2, 3.7% and 3.8% for Breakfast and 50 Salads and 4.2% and 1.0% for Breakfast and MPII Cooking 2. It is expected that MPII Cooking 2, being the largest dataset, would benefit less from cross-dataset training than the two smaller datasets. Qualitative results are shown in Fig. 6, comparing DDLSTM to the second best performing LSTM architecture in each case.

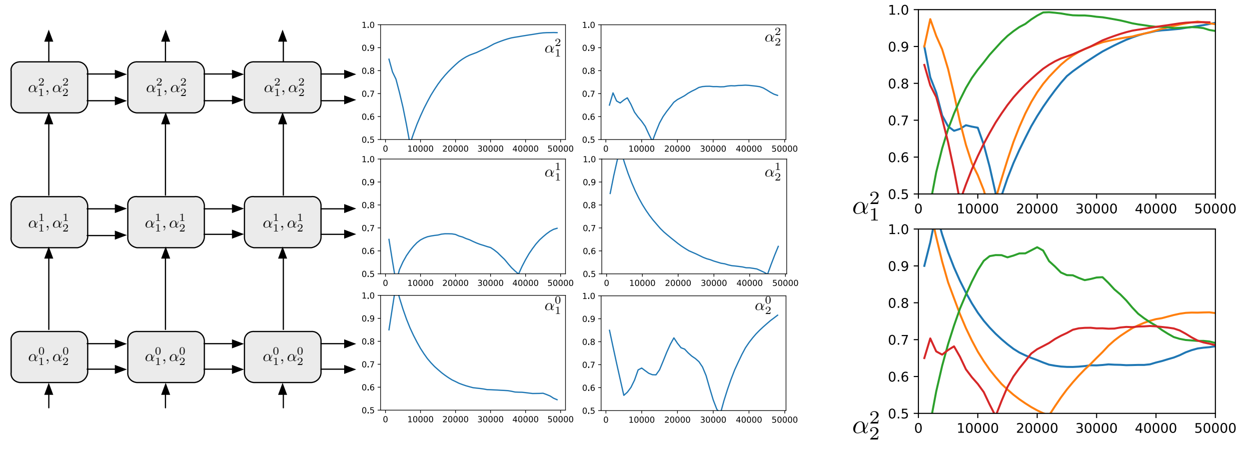


Figure 7: Examples of α progress during training. The graphs on the right show how α's perform with different initialisations.

D1	D2	LSTM Layers	D1 Avg	D2 Avg
50 Salads	MPII Cooking 2	1	45.7	41.3
50 Salads	MPII Cooking 2	2	46.6	41.0
50 Salads	MPII Cooking 2	3	**47.1**	**41.5**
50 Salads	MPII Cooking 2	4	46.1	40.6
50 Salads	MPII Cooking 2	5	41.0	40.3
50 Salads	MPII Cooking 2	10*	39.7	38.0
Breakfast	50 Salads	1	28.4	45.0
Breakfast	50 Salads	2	29.0	45.2
Breakfast	50 Salads	3	**29.1**	**46.3**
Breakfast	50 Salads	4	28.6	44.3
Breakfast	50 Salads	5	29.0	46.0
Breakfast	50 Salads	10*	25.5	40.7
Breakfast	MPII Cooking 2	1	29.2	**40.8**
Breakfast	MPII Cooking 2	2	29.9	40.1
Breakfast	MPII Cooking 2	3	**30.5**	40.1
Breakfast	MPII Cooking 2	4	**30.5**	38.8
Breakfast	MPII Cooking 2	5	30.3	38.4
Breakfast	MPII Cooking 2	10*	28.1	35.6

Table 2: Average accuracies over all dataset splits, comparing DDLSTM architecture with different depths 1-10. *Depth 10 used a history of 100 due to memory constraints.

5.2. Discussion

Table 2 gives an evaluation of how the number of levels (or depth) of the DDLSTM architecture affects performance. In general, there is a marginal improvement as the number of levels increases up to 3 with a drop off afterwards, although we observed one case where there was a slight drop (MPII Cooking 2 when trained with Breakfast).

Fig. 7 shows an example of how the α values, which control the dual-domain cross-contamination, change during training. In [2], automatic domain alignment layers were shown to use more cross-contamination for high-level layers than low-level layers. We sometimes observed analogous behaviour, such as α_1^0, α_1^1 and α_1^2. Here, $\alpha_1^0 \approx 0.5$ indicates no cross-contamination, while $\alpha_1^2 \approx 1$ indicates total cross-contamination. However, other behaviours were also seen, such as α_2^0, α_2^1 and α_2^2. One possible explanation is that the DDLSTMs are already being fed high-level features, so there is a less drastic deep-to-shallow progression in terms of what these features represent. This could also be a reason for there being a small, rather than large, improvement when increasing the depth of the DDLSTM network (Table 2).

We noted earlier that the method is robust to initialisations of α values. Fig. 7 also shows how different initialisations converge to similar α values after training. We note that α values from multiple runs are unlikely to be identical, given that batches contain random sample orderings.

5.3. Comparison to Published Results

Very few works have attempted online frame-level accuracy on these datasets, with most works focusing on offline classification [16, 37, 25, 19, 38]. To compare to these, we also evaluate DDLSTM with look-ahead, i.e. allowing it to see half the training history size into the future when classifying the current frame.

In Table 3, we report results on Breakfast, showing our method outperforms published results operating online - note that [7] only provides results on a single split in the dataset. We use "offline multi-pass" to refer to methods that iteratively optimise the classification of frames, by multi-pass segmentation of the whole video. "Offline single pass" methods have access to future frames, but only as a single pass, e.g. bi-directional LSTM. "Online" methods classify each frame as soon as it is seen, with no access to future frames, e.g. uni-directional LSTM. We do not expect our single-pass results to outperform multi-pass offline evaluations, but provide these results for completion.

In Table 4, we compare to published offline results on 50 Salads, using publicly available features from [25]. These use mid-level classes (17 classes plus a background

Method	Mode	Accuracy
[22]	offline multi-pass	56.3
[35]	offline multi-pass	43.0
DDLSTM (look-ahead) w/f [22]	offline single-pass	26.4
DDLSTM (look ahead)	offline single-pass	32.6
[35]	online	27.2
[7] (only evaluated on 1/4 splits)	online	32.6
[32]	online	28.5
DDLSTM w/f [22]	online	23.8
DDLSTM	online	29.1

Table 3: Comparative analysis on the Breakfast dataset, using our features as well as with features (w/f) from [22]. The DDLSTM uses the public features from 50 salads [25] (used in Table 4) as its second domain.

Method	Mode	Accuracy
Bi-LSTM (1 layer) [25]	offline single-pass	55.7
ED-TCN [25]	offline single-pass	64.7
DDLSTM (look-ahead) w/f [25]	offline single-pass	60.9
LSTM w/f [25]	online	57.6
DDLSTM w/f [25]	online	59.1

Table 4: Comparative analysis on the 50 salads dataset, using mid-level classes with features (w/f) from [25]. The DDLSTM uses the public features from Breakfast [22] as its second domain (used in Table 3).

class), which is significantly easier than what we report in Table 1 where we use all 52 lowest-level classes. Our online results and results with look ahead are only slightly worse than others evaluated offline. We have not found any published results for online performance on 50 salads mid-level, or any other results using publicly available features.

5.4. Effect of Related Domain Adaptation

To investigate how much of the DDLSTM's improvement comes from exploiting related temporal information, we evaluated whether 50 Salads (split 0) benefits equally from three large-scale datasets with various levels of domain relatedness. We test on THUMOS [17] (features from [10]), ActivityNet [15] (features from [15]) and EPIC Kitchens [6] (ImageNet ResNet 50 features). Of these, only EPIC presents a related 'kitchen' domain. THUMOS and ActivityNet capture actions unrelated to the kitchen domain and contain very few actions per sequence.

We report results in Table 5. Only by fine-tuning, results show that the related dataset EPIC provides best performance. We observe no benefit to in jointly training using DDLSTM with THUMOS or ActivityNet. We however observe clear improvements when jointly training with EPIC Kitchens, for both EPIC (by 1.6%) and 50 Salads (by 4%). We conclude that 1) DDLSTM is particularly suited for related domains, and that 2) a higher increase in accuracy is expected for smaller datasets as these leverage missing temporal knowledge from the larger dataset.

D1	D2	Training	LSTM Type	D1 Acc	D2 Acc
ActivityNet	50 Salads	Pt/ft	LSTM	44.4	42.1
ActivityNet	50 Salads	Joint	DDLSTM	44.3	42.2
Thumos	50 Salads	Pt/ft	LSTM	65.9	42.0
Thumos	50 Salads	Joint	DDLSTM	66.1	42.3
EPIC	50 Salads	Pt/ft	LSTM	31.5	**44.9**
EPIC	50 Salads	Joint	DDLSTM	33.1	**48.9**

Table 5: Classification accuracy when learning 50 salads (split 0) from larger datasets. Pt/ft refers to results on D1 when pre-training on D2 and fine tuning on D1 and vice versa. Figure shows that the large dataset with related domain (EPIC) performs better for pre-training, and shows larger improvement using DDLSTM.

6. Conclusion and Future Work

In this paper, the Dual-Domain LSTM (DDLSTM) was introduced, which is capable of learning temporal information from two domains at the same time. Given batches consisting of samples from both domains, DDLSTM applies a dual-domain batch normalisation on both input-to-hidden and hidden-to-hidden LSTM weights. This calculates separate batch statistics for each domain, but learns a parameter which determines how much cross-contamination between domains should be included in a fully differentiable manner. The learnt parameters are shared across timesteps, but the batch normalisation calculation is performed on data at each timestep separately. We evaluated the DDLSTM architecture on online action recognition, using three cooking datasets with multiple actions per video and framewise labels. DDLSTM was found to outperform (by a 3.5% average across all datasets) the standard LSTM (both jointly trained, and pre-trained and fine-tuned) and the batch-normalised LSTM upon which it builds.

This paper presents a number of opportunities (A-D) for future investigation. A) Learning from more than two related datasets/domains at the same time. This would require modifications to the contribution functions (Equations 5 and 6) and an increase in the number of αs to $d(d-1)l$, where d and l are the number of datasets and LSTM layers. B) Automatically adjusting the makeup of each batch could provide performance improvements, in a similar fashion to the way α values are leaned for cross-contamination. C) Using an attention mechanism [34] to determine which items within a batch are most useful for cross-contamination. D) Incorporating frame-based domain adaptation methods into the feature extractor as well as the LSTM.

Acknowledgement: Research supported by EPSRC LOCATE (EP/N033779/1) and uses publicly available data.

References

[1] P. P. Busto and J. Gall. Open Set Domain Adaptation. In *International Conference on Computer Vision*, 2017. 1

[2] F. M. Carlucci, L. Porzi, B. Caputo, E. Ricci, and S. R. Bulò. AutoDIAL: Automatic DomaIn Alignment Layers. In *International Conference on Computer Vision*, 2017. 1, 2, 4, 7

[3] J. Carreira and A. Zisserman. Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset. In *Computer Vision and Pattern Recognition*, 2017. 2, 5

[4] T. Cooijmans, N. Ballas, C. Laurent, Ç. Gülçehre, and A. Courville. Recurrent Batch Normalization. In *arXiv, 1603.09025*, 2016. 1, 2, 3, 4, 6

[5] G. Csurka, F. Baradel, and B. Chidlovskii. Discrepancy-Based Networks for Unsupervised Domain Adaptation : A Comparative Study. In *International Conference on Computer Vision*, 2017. 2

[6] D. Damen, H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, and M. Wray. Scaling egocentric vision: The epic-kitchens dataset. In *European Conference on Computer Vision*, 2018. 1, 8

[7] R. De Geest and T. Tuytelaars. Modeling temporal structure with LSTM for online action detection. In *Winter Conference on Applications of Computer Vision*, 2018. 7, 8

[8] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell. Long-Term Recurrent Convolutional Networks for Visual Recognition and Description. *Transactions on Pattern Analysis and Machine Intelligence*, 39(4):677–691, 2017. 1

[9] Y. Du, W. Wang, and L. Wang. Hierarchical Recurrent Neural Network for Skeleton Based Action Recognition. In *Computer Vision and Pattern Recognition*, 2015. 1

[10] J. Gao, Z. Yang, and R. Nevatia. Cascaded boundary regression for temporal action detection. In *British Machine Vision Conference*, 2017. 8

[11] B. Gong, Y. Shi, F. Sha, and K. Grauman. Geodesic Flow Kernel for Unsupervised Domain Adaptation. In *Computer Vision and Pattern Recognition*, 2012. 2

[12] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber. LSTM: A Search Space Odyssey. *Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, 2017. 2

[13] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. J. Smola. A Kernel Method for the Two-Sample-Problem. In *Advances in Neural Information Processing Systems*, 2006. 1

[14] P. Haeusser, T. Frerix, A. Mordvintsev, and D. Cremers. Associative Domain Adaptation. In *International Conference on Computer Vision*, 2017. 2

[15] F. C. Heilbron, V. Escorcia, B. Ghanem, and J. C. Niebles. ActivityNet: A Large-Scale Video Benchmark for Human Activity Understanding. In *Computer Vision and Pattern Recognition*, 2015. 5, 8

[16] D. A. Huang, L. Fei-Fei, and J. C. Niebles. Connectionist temporal modeling for weakly supervised action labeling. In *European Conference on Computer Vision*, 2016. 7

[17] H. Idrees, A. R. Zamir, Y. G. Jiang, A. Gorban, I. Laptev, R. Sukthankar, and M. Shah. The THUMOS challenge on action recognition for videos in the wild. *Computer Vision and Image Understanding*, 155:1–23, 2017. 5, 8

[18] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, 2015. 2

[19] S. Jiao. Global for Coarse and Part for Fine: A Hierarchical Action Recognition Framework. *International Conference on Image Processing*, pages 2630–2634, 2018. 7

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances In Neural Information Processing Systems*, 2012. 2

[21] H. Kuehne, A. Arslan, and T. Serre. The Language of Actions: Recovering the Syntax and Semantics of Goal-Directed Human Activities. In *Computer Vision and Pattern Recognition*, 2014. 1, 5

[22] H. Kuehne, J. Gall, and T. Serre. An end-to-end generative framework for video segmentation and recognition. In *Winter Conference on Applications of Computer Vision*, 2016. 8

[23] H. Kuehne, T. Serre, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: A Large Video Database for Human Motion Recognition. In *International Conference on Computer Vision*, 2011. 5, 6

[24] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio. Batch Normalized Recurrent Neural Networks. In *International Conference on Acoustics, Speech and Signal Processing*, 2016. 2

[25] C. Lea, M. D. F. Ren, A. Reiter, and G. D. Hager. Temporal Convolutional Networks for Action Segmenta-

tion and Detection. In *Computer Vision and Pattern Recognition*, 2017. 7, 8

[26] Y. Li, C. Lan, J. Xing, W. Zeng, C. Yuan, and J. Liu. Online Human Action Detection using Joint Classification-Regression Recurrent Neural Networks. In *European Conference on Computer Vision*, 2016. 6

[27] J. Liu, G. Wang, L. Y. Duan, K. Abdiyeva, and A. C. Kot. Skeleton-Based Human Action Recognition with Global Context-Aware Attention LSTM Networks. *Transactions on Image Processing*, 27(4):1586–1599, 2018. 1

[28] M. Long, J. Wang, G. Ding, J. Sun, and P. S. Yu. Transfer Feature Learning with Joint Distribution Adaptation. In *Computer Vision and Pattern Recognition*, 2013. 1

[29] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch Networks for Multi-task Learning. In *Computer Vision and Pattern Recognition*, 2016. 2

[30] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond Short Snippets: Deep Networks for Video Classification. In *Computer Vision and Pattern Recognition*, 2015. 2

[31] W. Nie, A. Liu, J. Yu, Y. Su, L. Chaisorn, Y. Wang, and M. S. Kankanhalli. Multi-View Action Recognition by Cross-Domain Learning. In *International Workshop on Multimedia Signal Processing*, 2014. 2

[32] T. Perrett and D. Damen. Recurrent Assistance : Cross-Dataset Training of LSTMs on Kitchen Tasks. In *International Conference on Computer Vision: Workshop on Assistive Computer Vision and Robotics*, 2017. 1, 8

[33] T. Perrett and M. Mirmehdi. Cost-Based Feature Transfer for Vehicle Occupant Classification. In *Asian Conference on Computer Vision Workshop on Technology for Smart Vehicles*, 2013. 1

[34] Y. Qin, D. Song, H. Cheng, W. Cheng, G. Jiang, and G. W. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. In *International Joint Conference on Artificial Intelligence*, 2017. 8

[35] A. Richard, H. Kuehne, A. Iqbal, and J. Gall. NeuralNetwork-Viterbi: A framework for weakly supervised video learning. In *Computer Vision and Pattern Recognition*, 2018. 8

[36] M. Rohrbach, S. Amin, M. Andriluka, and B. Schiele. A Database for Fine Grained Activity Detection of Cooking Activities. In *Computer Vision and Pattern Recognition*, 2012. 1, 5

[37] B. Singh, T. K. Marks, M. Jones, O. Tuzel, and M. Shao. A Multi-stream Bi-directional Recurrent Neural Network for Fine-Grained Action Detection. In *Computer Vision and Pattern Recognition*, 2016. 7

[38] H. Song, X. Wu, B. Zhu, Y. Wu, M. Chen, and Y. Jia. Temporal Action Localization in Untrimmed Videos using Action Pattern Trees. In *Computer Vision and Pattern Recognition*, 2018. 7

[39] K. Soomro, A. R. Zamir, and M. Shah. UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild. *arXiv 1212.0402*, 2012. 5

[40] S. Stein and S. J. McKenna. Combining Embedded Accelerometers with Computer Vision for Recognizing Food Preparation Activities. In *International Joint Conference on Pervasive and Ubiquitous Computing*, 2013. 1, 5

[41] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. In *Computer Vision and Pattern Recognition*, 2016. 6

[42] H. Venkateswara, J. Eusebio, S. Chakraborty, and S. Panchanathan. Deep Hashing Network for Unsupervised Domain Adaptation. In *Computer Vision and Pattern Recognition*, 2017. 2

[43] Z. Wu, Y.-G. Jiang, J. Wang, J. Pu, and X. Xue. Exploring Inter-feature and Inter-class Relationships with Deep Neural Networks for Video Classification. In *ACM International Conference on Multimedia*, 2014. 2

[44] V. W. Zheng, D. H. Hu, and Q. Yang. Cross-Domain Activity Recognition. In *International Conference on Ubiquitous Computing*, 2009. 2