ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/318329915

Deep neural networks for kitchen activity recognition

Conference Paper · May 2017

DOI: 10.1109/IJCNN.2017.7966102

CITATIONS
5

READS
116

4 authors:

Juarez Monteiro
Pontifícia Universidade Católica do Rio Grande do Sul
11 PUBLICATIONS **21** CITATIONS
SEE PROFILE

Roger Granada
Pontifícia Universidade Católica do Rio Grande do Sul
33 PUBLICATIONS **181** CITATIONS
SEE PROFILE

Rodrigo C. Barros
Pontifícia Universidade Católica do Rio Grande do Sul
97 PUBLICATIONS **1,173** CITATIONS
SEE PROFILE

Felipe Meneguzzi
Pontifícia Universidade Católica do Rio Grande do Sul
149 PUBLICATIONS **852** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Clustering Data Streams View project

Project Hierarchical Multi-Label Classification View project

All content following this page was uploaded by Rodrigo C. Barros on 23 April 2018.

The user has requested enhancement of the downloaded file.

Deep Neural Networks for Kitchen Activity Recognition

Juarez Monteiro*, Roger Granada*, Rodrigo C. Barros†, and Felipe Meneguzzi†
Faculdade de Informática
Pontifícia Universidade Católica do Rio Grande do Sul
Av. Ipiranga, 6681, 90619-900, Porto Alegre, RS, Brazil
* Email: {juarez.santos, roger.granada}@acad.pucrs.br
† Email: {rodrigo.barros, felipe.meneguzzi}@pucrs.br

***Abstract*—With the growth of video content produced by mobile cameras and surveillance systems, an increasing amount of data is becoming available and can be used for a variety of applications such as video surveillance, smart homes, smart cities, and in-home elder monitoring. Such applications focus in recognizing human activities in order to perform different tasks allowing the opportunity to support people in their different scenarios. In this paper we propose a deep neural architecture for kitchen human action recognition. This architecture contains an ensemble of convolutional neural networks connected through different fusion methods to predict the label of each action. Experiments show that our architecture achieves the novel state-of-the-art for identifying cooking actions in a well-known kitchen dataset.**

I. Introduction

Effective assistive applications require accurate identification of the activities that are performed by the user being helped. Here, activity recognition refers to the task of dealing with noisy low-level data directly from sensors [1]. Such task is particularly challenging in the real physical world, since it either involves fusing information from a number of sensors or inferring enough information using a single sensor. Failure to correctly identifying the activity the user is performing has a cascade effect that often leads to users being frustrated and giving up using the assistive application.

Single-sensor activity recognition often relies on a video camera feed [2], which has posed a challenging research problem in computer vision and machine learning. Advances in hardware and greater availability of data have allowed deep learning algorithms, and Convolutional Neural Networks (CNNs) [3] in particular, to consistently improve on the state-of-the-art. CNNs achieve state-of-the-art results when dealing with image-based tasks such as object recognition, detection, and semantic segmentation [4], [5]. Encouraged by those results, more and more applications are relying on deep neural architectures to perform video-based tasks [2].

In this paper, we address the problem of recognizing human activities in an indoor environment with a single static camera. Our main contribution is on supporting people when they are in the kitchen, with the final goal of recognizing their actions when cooking meals. Our approach relies on a deep neural architecture that comprises multiple convolutional neural networks that are fused prior to performing the action classification. We perform experiments using the Kitchen Scene Context based Gesture Recognition dataset (KSCGR) [6], and we show that our proposed approach outperforms the current state-of-the-art method [7] for this particular dataset.

This paper is organized as follows. Section II details our novel deep neural architecture for action recognition, whereas Section III presents a thorough experimental analysis for assessing the performance of our proposed approach. Section IV points to related work and we finish this paper with our conclusions and future work directions in Section V.

II. Architecture Design

Machine learning algorithms such as artificial neural networks (ANN) have been used to address many challenges of action and activity recognition. For decades, building machine learning systems required considerable domain expertise to create an internal representation (feature construction [8]) from which the learning subsystem could detect or classify patterns within the input. Deep learning approaches such as convolutional neural networks mitigate this problem by automatically learning representations in terms of hierarchical features, allowing the computer to build complex concepts out of simpler concepts. In this paper, we develop a deep neural architecture for action recognition in indoor environments with a fixed camera using an ensemble of convolutional neural networks (CNNs). Four different fusion methods including a support vector machine classifier (SVM) [9] and a long short-term memory network (LSTM) [10] are used to fuse the output of the CNNs and provide the final prediction of the input frame. Our architecture has three main components: i) data pre-processing, ii) convolutional networks for action recognition, and iii) fusion strategies for final classification.

Figure 1 illustrates the pipeline of our architecture where *RGB* represents the pre-processed dataset with RGB video frames; *OFL* represents the pre-processed dataset generated by dense optical flow; *AlexNet*, *GoogLeNet*, and *SqueezeNet* are the convolutional neural network architectures we use to recognize activities; *NN* is a neural network that weights the contribution of the probabilities generated by the output of the previous CNNs; *Mean* computes the arithmetic mean of the probabilities provided by the CNNs; *SVM* is a support vector machine classifier with linear kernel that classifies the

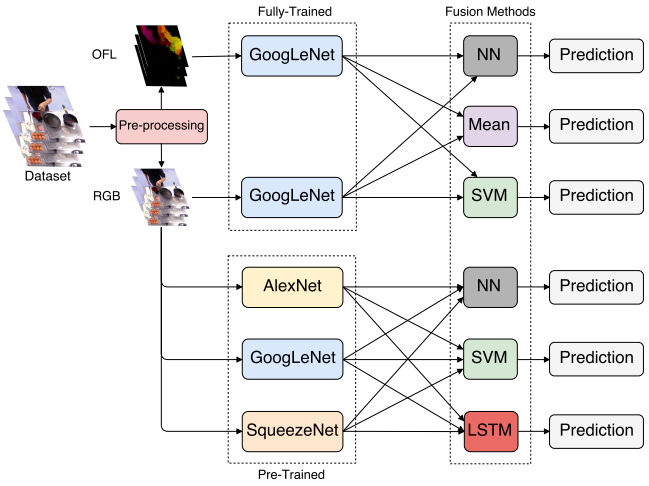

Fig. 1. Pipeline of our architecture for action recognition.

probability vectors from the CNNs; and *LSTM* is a recurrent neural network architecture that fuses the output probability vectors from the CNNs to provide the final classification.

We separate the convolutional neural network architectures into two groups: the pre-trained CNNs and the fully-trained CNNs. The pre-trained group contains 3 neural networks that were pre-trained on the ImageNet data [11]. We use the pre-trained AlexNet [4], GoogLeNet [12], and SqueezeNet [13] models freely-available in the Caffe Model Zoo repository[1]. The fully-trained group contains a single neural network that is trained from scratch in the well-known kitchen dataset KSCGR [6].

The pipeline of our architecture receives images from the kitchen dataset as input for pre-processing. Pre-processing extracts dense optical flow representations from the input images and resizes all images to 256×256, generating two new input data hereafter called *OFL* for images with dense optical flow and *RGB* for the original RGB data. The system feeds the pre-trained and fully-trained networks with the RGB and OFL data, generating output vectors that indicate the probability an image has of belonging to each class. Each fusion method (*NN*, *Mean*, *SVM* and *LSTM*) receives the concatenation of the probability vectors from the CNNs and predicts the final class of the input image. In what follows, we further detail each component of the proposed architecture.

A. Data pre-processing

Pre-processing consists of two steps: image resizing and optical flow generation. Resizing is important since it reduces the multidimensional space required by the CNNs to learn suitable features for image classification, as well as the total processing time. This step resizes all images of the dataset to a fixed resolution of 256×256. The second step generates the dense optical flow representation [14] of adjacent frames. In a nutshell, optical flow represents the 2D displacement of pixels between frames generating vectors corresponding to the movement of points from the first frame to the second. Dense optical flow generates these displacement vectors, *i.e.*, for both horizontal and vertical displacements, regarding all points within frames. In order to generate the final image for each sequence of frames, we combine the 2-channel optical flow vectors and associate color to their magnitude and direction. Magnitudes are represented by colors and directions through hue values. The output of the data pre-processing step consists of two datasets containing the original data with RGB channels and resized size (*RGB*), and the optical flow data that encapsulates motion across frames (*OFL*).

[1]https://github.com/BVLC/caffe/wiki/Model-Zoo

B. CNN Architectures

In this work, we divided the convolutional neural networks into two groups: fully-trained and pre-trained networks. The fully-trained networks have the same architecture and training hyper-parameters, and they are trained from scratch receiving the two streams of data (RGB and OFL). The network trained on RGB is hereafter called $\text{GoogLeNet}_{[\text{RGB}]}$, whereas the network trained on OFL is called $\text{GoogLeNet}_{[\text{OFL}]}$. Both architectures are 22-layer deep and their inception modules contain convolutional filters in different scales/resolutions, covering clusters of diverse information. Each network receives video frames as input, which traverse several convolutional layers, pooling layers, and fully-connected layers (FC). After a *Softmax* layer, the network outputs a vector containing the probability each frame has of belonging to each class.

Even though a number of off-the-shelf CNN architectures are available [15], [2], in this work we make use of three pre-trained networks. We choose an architecture based on inception modules [12] due to its reasonable performance and reduced number of trainable parameters, hereafter called $\text{GoogLeNet}_{[\text{off-the-shelf}]}$ and $\text{GoogLeNet}_{[\text{Fine-tuned}]}$. The other two architectures are based on AlexNet [4] (hereafter called $\text{AlexNet}_{[\text{Fine-tuned}]}$) and SqueezeNet [13] (herafter called $\text{SqueezeNet}_{[\text{Fine-tuned}]}$), due to their reduced number of layers and parameters. $\text{AlexNet}_{[\text{Fine-tuned}]}$, $\text{GoogLeNet}_{[\text{off-the-shelf}]}$, $\text{GoogLeNet}_{[\text{Fine-tuned}]}$, and $\text{SqueezeNet}_{[\text{Fine-tuned}]}$ were pre trained on the 1.3-million-image ILSVRC 2012 ImageNet dataset [11]. Despite the fact that the AlexNet model provided in Caffe Zoo reposity has some small differences from the original AlexNet by Krizhevsky et al. [4], we do not believe our results would significantly change due to small architectural and optimization modifications. Similarly to the fully-trained networks, after a *Softmax* layer each network outputs a vector with the probability of the input image for each class. The difference between $\text{GoogLeNet}_{[\text{off-the-shelf}]}$ and $\text{GoogLeNet}_{[\text{Fine-tuned}]}$ relies on the fact that in the former we adjust the last layer to the number of classes of our dataset and "freeze" the remaining layers during training, *i.e.*, we do not update weights of any layer but the last. In fine-tuned networks ($\text{AlexNet}_{[\text{Fine-tuned}]}$, $\text{GoogLeNet}_{[\text{Fine-tuned}]}$, and $\text{SqueezeNet}_{[\text{Fine-tuned}]}$), we update all pre-trained layers with different learning rates, allowing the network to learn features

more specific to the target dataset, while starting from a consistent set of weights.

The idea behind our architecture is that distinct networks may capture different data patterns. In addition, different views from the same data may also help in classifying frames into actions. Thus, the same network processes data with different representations (RGB and OFL), and three different networks (AlexNet, GoogLeNet and SqueezeNet) process the same data (RGB).

C. Fusion Methods

Since the output of each CNN is a vector containing the probability scores for each class, our model architecture allows for the application of distinct fusion methods for providing the ultimate classification. The fusion methods intend to merge these vectors in order to increase the accuracy for the action recognition task. Before fusing probabilities, we merge the output of the pre-trained networks $\text{GoogLeNet}_{[\text{RGB}]}$ and $\text{GoogLeNet}_{[\text{OFL}]}$, generating the $\text{GoogLeNet}_{[\text{RGB+OFL}]}$ vector. We employ a similar strategy to the fully-trained networks $\text{AlexNet}_{[\text{Fine-tuned}]}$, $\text{GoogLeNet}_{[\text{Fine-tuned}]}$, and $\text{SqueezeNet}_{[\text{Fine-tuned}]}$, by generating the *3CNNs* vector. The new merged vectors are used as input to the fusion methods in order to generate predictions for each class. Figure 1 shows our four different approaches: i) a neural network (*NN*) that weights the contribution of the probability vectors, ii) the standard arithmetic mean, *i.e.*, weight 0.5 for both vectors (*Mean*), iii) a multi-class linear Support Vector Machine (*SVM*) [9], and iv) a special case of recurrent neural network called long short-term memory (LSTM) [10].

The *NN* fusion contains a single-layer neural network to optimize the weights of the probabilities derived from the output of the CNNs. Figure 2 illustrates the structure of such network when using RGB and OFL data, where w_1 and w_2 are learned weights, $[A]$ is the vector containing the probabilities from the output of the CNN that processes the OFL images, $[B]$ is the vector containing the probabilities for each class generated by the output of the CNN that processes the RGB images, and $[C]$ is the vector containing the weighted mean for each class. The idea behind this neural network is that its weights (w_1 and w_2) can be learned automatically by minimizing a loss function and backpropagating the gradients. During test time, this fusion method employs the learned parameters to properly weight the contribution of each merged vector. The *Mean* fusion receives the output vector from both RBG and OFL CNNs and calculates the arithmetic mean for each class (equal weights), assigning to the image the class with the highest score. The *SVM* fusion is based on a multi-class linear Support Vector Machine trained with the output of the CNNs when using the validation data.At test time, the SVM predicts the class with the largest score. The *LSTM* fusion contains a recurrent neural network in the form of a chain of repeating modules of weights that intends to learn long-term dependencies. These long-term dependencies are represented in the form of previous information connected to the present image, *e.g.*, the class of the current image is represented not only by the information of the current frame, but also by the information extracted from previous frames. LSTM units have hidden state augmented with nonlinear mechanisms to allow states to propagate without modification, be updated, or be reset using simple learned gating functions [16].

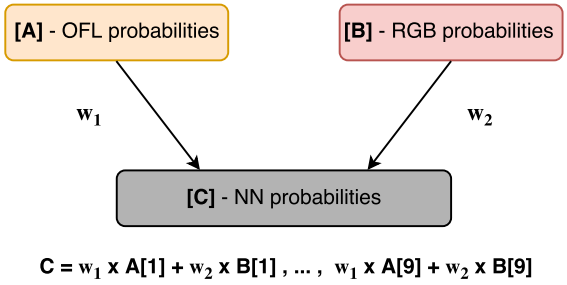

Fig. 2. Single-layer neural network developed to compute the optimal weighted average from the outputs of the convolutional neural networks.

D. Post processing

Since the process of identifying actions occurs frame by frame instead of the entire video, sometimes the misclassification of a small number of frames of an action may occur. Since an activity does not occur in a single frame or in a very small number of frames, we believe that a frame in the middle of a sequence of 20 frames that contains a different class probably suggests that the frame was misclassified. For example, the misclassification of 5 frames of the *Baking* action in the middle of ≈ 200 frames of the *None* action. Following the work of Bansal *et al.* [7], we apply a smoothing process on the output sequence of classes in order to identify and fix frames that are probably incorrectly-classified. This smoothing process consists of sliding a window of fixed-size through the temporally sorted predicted classes assigning to the target frame (the frame in the center of the window) the majority voting of all frames within the window.

III. Experimental Analysis

In this section, we describe the dataset used in our experiments for indoor fixed-camera action recognition, the implementation details regarding the CNNs and fusion methods, and the results that were achieved by our approach in comparison with the current state-of-the-art.

A. KSCGR Dataset

The Kitchen Scene Context based Gesture Recognition dataset[2] (KSCGR)[6] is a fine-grained kitchen action dataset released as a challenge in ICPR 2012[3]. The dataset contains scenes captured by a kinect sensor fixed on the top of the kitchen, providing synchronized color and depth image sequences. Each video is 5 to 10 minutes long, containing 9,000

[2]http://www.murase.m.is.nagoya-u.ac.jp/KSCGR/
[3]http://www.icpr2012.org/

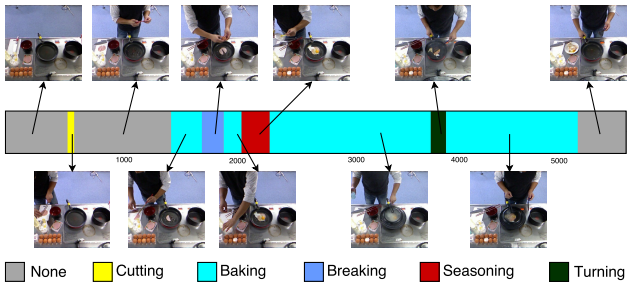

Fig. 3. Example of the frame/action sequence for the "ham and egg" menu.

to 18,000 frames. The organizers of the dataset assigned labels to each frame indicating the type of gesture performed by the actors. There are 8 cooking gestures in the dataset: *breaking*, *mixing*, *baking*, *turning*, *cutting*, *boiling*, *seasoning*, *peeling*, and *none*, where *none* means that there is no action being performed in the current frame. These gestures are performed in five different menus for cooking eggs in Japan: *ham and eggs*, *omelet*, *scrambled egg*, *boiled egg*, and *kinshi-tamago*. A total of 7 different subjects perform each menu. The ground truth data contains the frame id and the action being performed within the frame.

We divided the dataset into training, validation, and test sets. The training set contains 4 subjects, each of them performing 5 recipes, *i.e.*, 20 videos and 139,196 frames in total. We use the validation set to obtain the model configuration that performs best, *i.e.*, the configuration with the highest accuracy. This set contains 1 subject performing 5 recipes with 32,897 frames in total. We use the test set to assess the accuracy of the selected model in unseen data. This set contains 2 subjects, each performing 5 recipes, *i.e.*, 10 videos with 55,781 frames in total. Figure 3 shows the sequence of frames and actions when performing the menu *Ham and Egg*, where the colored bar represents the timeline of appearance of frames and actions, and the images illustrate examples of each action performed in the video.

B. Implementation

Fully-trained CNNs architecture: in order to perform the action recognition task we use an inception-based CNN architecture [12] trained from scratch in RGB and OFL separately. The training phase uses mini-batch stochastic gradient with momentum (0.9). For each iteration, the network forwards a mini-batch of 128 samples. We apply data augmentation with random crops, *i.e.*, a different crop in a randomly selected part of the image is selected, as well as a probabilistic horizontal flip, generating a sub-image of 224×224. All images have their pixels subtracted by the mean pixel values of all training images. All convolutions, including those within the inception modules, use rectified linear activation units (ReLU). Regarding weight initialization, we employ the *Xavier* algorithm that automatically determines the value of initialization based on the number of input neurons. To minimize the chances of overfitting, we apply dropout on the fully-connected layers with a probability of 70%. The learning rate is set to 10^{-3} and we drop it by a factor of 50 every epoch, stopping the training after 43.5k iterations (30 epochs).

Pre-trained CNNs architectures: all networks of this group were pre-trained over the ILSVRC 2012 ImageNet dataset [11]. For the training phase, we kept almost the same configuration for all networks, using a mini-batch of 128 samples with a random crop of 224×224 as well as random horizontal flip. Each image has its pixels subtracted by the mean value of pixels of each channel. During training, we freeze all but the last layer of GoogLeNet$_{\text{[off-the-shelf]}}$, performing the weights and bias updates only for the last fully-connected layer for 10 epochs, increasing the learning rate of the layer by 10 (setting learning rate of the weights to 10 and learning rate of the bias to 20). For fine-tuned models (AlexNet$_{\text{[Fine-tuned]}}$, GoogLeNet$_{\text{[Fine-tuned]}}$, and SqueezeNet$_{\text{[Fine-tuned]}}$), we update all weights but with a different learning rate for the last layer. We increase the learning rate of the weights in the last layer from 1 to 10 and the bias from 2 to 20, and decrease the global learning rate by 100. This configuration allows all layers to learn, though giving the final layer the capability to learn faster than the remaining layers.

NN: this fusion approach contains a neural network trained with data from the validation set for 10 epochs with weights w_1 and w_2 initialized with 0.5. We use the mean squared error loss function and optimize it through Adam [17] with a learning rate set to 10^{-3}.

SVM: we train the multi-class Support Vector Machine using the off-the-shelf implementation by Crammer and Singer [9] from *scikit-learn*[4] toolbox. Similarly to the neural network fusion, we train the SVM using the validation set. We use the linear kernel and default *scikit-learn* regularization parameter $C = 1$ with the square of the *hinge loss* as loss function.

LSTM: we implemented the long short-term memory using the Keras[5] neural networks library. Our configuration follows the implementation proposed by Donahue *et al.* [16] that connects a CNN with a LSTM, calling this model Long-term Recurrent Convolutional Network (LRCN). We explore various hyper-parameters using both training and validation sets, selecting the best architecture that contains 1024 hidden units with a dropout of 0.7 in order to avoid overfitting. We train and test the LSTM network in a sequence of 32 frames, and during training the stride is of 16 frames. We also apply the Adam [17] algorithm using a learning rate of 10^{-3}. We run the training phase for 30 epochs.

Post processing: the post processing consists of sliding a window of fixed-size through the predicted classes assigning to the target frame the majority voting of all frames within the window. In order to decide the size of the window, we used the predicted classes from the validation dataset. We performed several smoothing tests, varying the window-size from 10 to 50 increasing the step in 10 frames each time. Finally, we

[4]http://scikit-learn.org
[5]https://keras.io

TABLE I
PER-ACTIVITY ACCURACY IN THE KSCGR DATASET FOR ALL BASELINES AND FUSION METHODS.

Method	None	Breaking	Mixing	Baking	Turning	Cutting	Boiling	Seasoning	Peeling	Overall
$\text{GoogLeNet}_{[\text{RGB}]}$	0.644	0.275	0.289	0.671	**0.346**	0.588	0.287	0.363	0.117	0.689
$\text{GoogLeNet}_{[\text{OFL}]}$	0.519	0.341	0.314	0.600	0.194	0.545	0.128	0.382	0.449	0.631
$\text{GoogLeNet}_{[\text{RGB+OFL}]}$ + Mean	0.634	0.327	0.340	0.684	0.174	0.620	0.169	0.403	0.347	0.692
$\text{GoogLeNet}_{[\text{RGB+OFL}]}$ + SVM	0.679	0.357	0.432	0.689	0.000	0.526	0.444	0.601	0.455	0.721
$\text{GoogLeNet}_{[\text{RGB+OFL}]}$ + NN	0.690	0.354	0.452	0.693	0.012	0.516	**0.505**	0.651	0.382	0.726
$\text{GoogLeNet}_{[\text{Off-the-shelf}]}$	0.545	0.004	0.198	0.666	0.009	0.182	0.340	0.055	0.007	0.609
$\text{AlexNet}_{[\text{Fine-tuned}]}$	0.688	0.555	0.445	0.752	0.211	**0.636**	0.369	0.661	0.400	0.751
$\text{GoogLeNet}_{[\text{Fine-tuned}]}$	0.579	0.224	0.374	0.711	0.136	0.438	0.030	0.174	0.000	0.645
$\text{SqueezeNet}_{[\text{Fine-tuned}]}$	0.611	0.325	0.422	0.688	0.117	0.313	0.078	0.300	0.184	0.660
$\text{AlexNet}_{[\text{Fine-tuned}]}$ + SVM	0.636	0.570	0.395	0.741	0.173	0.447	0.303	0.520	0.335	0.717
$\text{GoogLeNet}_{[\text{Fine-tuned}]}$ + SVM	0.676	0.323	0.466	0.708	0.100	0.449	0.381	0.351	0.202	0.712
$\text{SqueezeNet}_{[\text{Fine-tuned}]}$ + SVM	0.538	0.211	0.240	0.593	0.028	0.010	0.078	0.113	0.013	0.587
3CNNs + SVM	0.604	0.363	0.449	0.678	0.105	0.269	0.165	0.236	0.042	0.667
3CNNs + NN	0.687	0.598	0.434	**0.757**	0.209	0.623	0.348	0.663	0.509	0.752
3CNNs + NN + PP	0.696	**0.621**	0.452	0.753	0.206	0.575	0.333	**0.725**	0.509	0.755
3CNNs + LSTM	0.737	0.508	0.536	0.739	0.191	0.571	0.458	0.416	0.738	0.775
3CNNs + LSTM + PP	**0.754**	0.504	**0.564**	0.749	0.190	0.560	0.469	0.384	**0.773**	**0.785**

chose the window size of 20 frames since it achieved the best accuracy results on validation data.

C. Results

In order to evaluate our approach, we compare the output of each fusion method in the test set. We use the classification generated by each individual CNN as baseline, and hence we can see whether the fusion method improves over each individual CNN. Table I shows the accuracy values for each class individually (*None*, *Breaking*, *Mixing*, *Baking*, *Turning*, *Cutting*, *Boiling*, *Seasoning*, *Peeling*), as well as the overall accuracy (*Overall*) that considers all classes at once.

We generate values of accuracy for the fully-trained models: GoogLeNet trained with RGB ($GoogLeNet_{[RGB]}$) and OFL ($GoogLeNet_{[OFL]}$) data, a merging of both networks $GoogLeNet_{[RGB+OFL]}$ with either the Mean ($GoogLeNet_{[RGB+OFL]}$+*Mean*), the SVM ($GoogLeNet_{[RGB+OFL]}$+*SVM*) or the neural network ($GoogLeNet_{[RGB+OFL]}$+*NN*) as the fusion method. For pre-trained models, we generate values of accuracy for an off-the-shelf network ($GoogLeNet_{[Off\text{-}the\text{-}shelf]}$), and for fine-tuned networks ($AlexNet_{[Fine\text{-}tuned]}$, $GoogLeNet_{[Fine\text{-}tuned]}$, and $SqueezeNet_{[Fine\text{-}tuned]}$). We test the fine-tuned models changing the output to a support vector machine classifier, generating $AlexNet_{[Fine\text{-}tuned]}$+*SVM*, $GoogLeNet_{[Fine\text{-}tuned]}$+*SVM* and $SqueezeNet_{[Fine\text{-}tuned]}$+*SVM*. The merging of the pre-trained networks (*3CNNs*) is also used as input to the fusion methods generating *3CNNs+SVM*, *3CNNs+NN*, and *3CNNs+LSTM*. Finally the post processing (*PP*) is applied on the output of the neural network generating *3CNNs+NN+PP* and *3CNNs+LSTM+PP*.

1) Overall Performance: As we can observe in Table I, the fusion of 3 pre-trained convolutional neural networks with a long short-term memory network and with the post processing strategy (*3CNNs+LSTM+PP*) achieves the best global accuracy (*All*) of 78.5%. The achieved results confirm our belief that different networks may identify different aspects (features) and their combination tends to improve results, as largely expected due to the *ensemble effect*. When comparing the merging of the 3 networks with single networks, *3CNN* achieves the best results for 6 (*None*, *Breaking*, *Mixing*, *Baking*, *Seasoning*, and *Peeling*) out of 9 actions. For *Cutting* and *Boiling*, our architecture using 3 networks achieves the second-best result. Our architecture did not perform well for the *Turning* action, and a possible reason for the low performance of the fusion methods for classifying *Turning* might be a mixture of the limited number of frames for this activity and the training phase using vector probabilities generated based on the validation set. Considering that our fusion methods are trained with predicted probabilities from validation data, any misclassification may lead to errors during the test phase.

2) Off-the-shelf vs. fully-trained vs. fine-tuned: In general, the $\text{GoogLeNet}_{[\text{Off-the-shelf}]}$ architecture is outperformed by its fully-trained version on the RGB data $GoogLeNet_{[RGB]}$ and by its fine-tuned version $\text{GoogLeNet}_{[\text{Fine-tuned}]}$. This result indicates that it is better to train the network from scratch when a large dataset is available or fine-tune the network allowing all layers to learn instead of simply learning the last layer. Comparing the network trained from scratch with the fine-tuned network, it seems better to train the network from scratch than to load pre-trained weights in ImageNet. These results may be explained by the fact that KSCGR's images are very different from ImageNet's.

3) Single vs. Merged vs. Merged/Fused : The merging of networks using different datasets ($GoogLeNet_{[RGB+OFL]}$) with a fusion method (*Mean*, *SVM* or *NN*) tends to improve results, achieving the maximum accuracy of $\approx 73\%$ when combining the network with a neural network fusion ($GoogLeNet_{[RGB+OFL]}$+*NN*). The use of trainable fusion methods ($GoogLeNet_{[RGB+OFL]}$+*SVM* and $GoogLeNet_{[RGB+OFL]}$+*NN*) decrease the accuracy of the *Turning* action probably because the validation data contains very few frames from this action. When comparing the fine-tuned networks with their versions with the SVM fusion, we can see that the original fine-tuned network achieves better results. $GoogLeNet_{[Fine\text{-}tuned]}$ + *SVM* is the only network

that achieves better results (7 out of 9 classes) when using a fusion algorithm. Virtually every result of the original fine-tuned versions of AlexNet and SqueezeNet are better than their versions using SVM. Comparing the three fine-tuned networks (AlexNet$_{\text{[Fine-tuned]}}$, GoogLeNet$_{\text{[Fine-tuned]}}$ and SqueezeNet$_{\text{[Fine-tuned]}}$), AlexNet$_{\text{[Fine-tuned]}}$ achieves the best results for 7 out of 9 classes and the best overall class score. These results indicate that a small network is capable of good performance probably because they properly avoid overfitting. Observing the 3 fusion methods, the SVM fusion achieves the worst results for most cases. *NN* and *LSTM* obtain similar results for most categories, and *3CNNs+LSTM* achieves the best overall classification with 77% of accuracy for all classes. Even though post processing may eliminate classes that contain a small number of frames, it seems its usage is quite beneficial since it consistently improves the obtained results. The post-processed versions achieve the best accuracy scores for 5 out of 9 classes when compared with all models, providing the best overall accuracy of 78.5%.

4) Unbalanced classes: Since classification accuracy takes into account only the proportion of correct results that a classifier achieves, it is not suitable for unbalanced datasets because it is biased towards classes with larger number of examples. Although other factors may change results, classes with a larger number of examples tend to achieve better results since the network has more examples to learn the variability of the features. By analyzing the KSCGR dataset, we note that it is indeed unbalanced, *i.e.*, classes are not equally distributed over frames. Figure 4 shows the distribution of accuracy scores over the classes for the GoogLeNet$_{\text{[RGB]}}$ network (left) and the distribution of these classes within the dataset. We can see that the dataset is unbalanced since the *None* action has the largest number of frames ($\approx$ 30% of the total) followed by *Baking* ($\approx$ 25% of the total), whereas *Breaking* contains only $\approx$ 3% of the frames. By checking the accuracy scores, we see that the GoogLeNet$_{\text{[RGB]}}$ achieves 28% of accuracy for the *Breaking* class and 67% of accuracy for the *Baking* class, meaning that the features that map to *Breaking* are not as evident as the features of *Baking*. The probable reason for this difference relies on the small number of training examples of the *Breaking* class that is passed to the network. *Baking*, on the other hand, is much more present within the dataset, improving the training experience and making the neural architecture generalize better for frames that belong to that class. For dealing with the unbalanced nature of the KSCGR dataset, we measure the performance of the fusion methods based on precision (P), recall (R), and F-Measure (F). Table II shows the values of precision, recall, F-measure, and accuracy achieved by the baselines (*GoogLeNet [RGB]* and *GoogLeNet [OFL]*), pre-trained, and fully trained networks. In order to compare with the current state-of-the-art on the KSCGR dataset, in Table II we present the performance achieved by Bansal *et al.* [7], which uses hand-crafted features (*HCF*) to identify activities, as well as their results after undergoing a similar post-processing step (*HCF+PP*). A large precision value means that the respective model can adjust very well to the features for identifying the class, whereas low values indicate that it cannot extract relevant features to identify the correct class among the remaining classes.

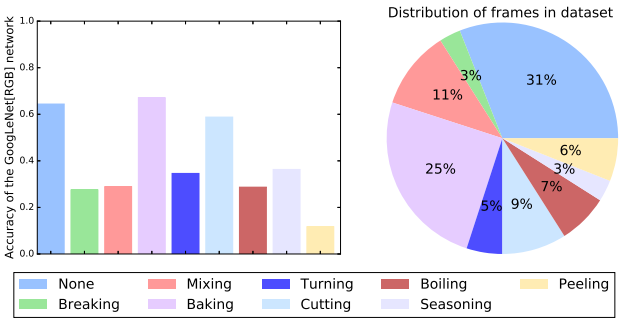

Fig. 4. Per-class accuracy and class distribution within the KSCGR dataset.

5) Our approach vs State-of-the-art: As we can observe in Table II, virtually all networks achieve similar values of precision, recall, and F-Measure. It is important to note that the combination of the 3 CNNs using the LSTM as fusion method and post processing (*3CNNs+LSTM+PP*) achieves the best scores for all measures. When compared with the hand-crafted features proposed by Bansal *et al.* [7], it is clear that our architecture provides better results. Only the network that was pre-trained on ImageNet and had all but the last layer frozen (GoogLeNet$_{\text{[Off-the-shelf]}}$) is outperformed by the (former) state-of-the-art.

6) Confusion Matrix: We also analyze the confusion matrix of the network that achieves the best performance without post processing since it is also important to see which classes that are commonly mistaken. The normalized confusion matrix depicted in Figure 5 shows the performance of the *3CNNs+LSTM* network, where rows represent the true classes

TABLE II
PRECISION, RECALL, F-MEASURE, AND ACCURACY FOR ALL BASELINES, FUSION METHODS AND THE (FORMER) STATE-OF-THE-ART APPROACH FOR THE KSCGR DATASET.

Approach	Precision	Recall	F-measure	Accuracy
HCF [7]	0.62	0.63	0.61	0.64
HCF + PP [7]	**0.68**	**0.68**	**0.68**	**0.72**
GoogLeNet$_{\text{[RGB]}}$	0.69	0.68	0.69	0.69
GoogLeNet$_{\text{[OFL]}}$	0.64	0.63	0.63	0.63
GoogLeNet$_{\text{[Off-the-shelf]}}$	0.61	0.61	0.61	0.61
GoogLeNet$_{\text{[RGB+OFL]}}$ + Mean	0.71	0.69	0.70	0.69
GoogLeNet$_{\text{[RGB+OFL]}}$ + SVM	0.67	0.72	0.70	0.72
GoogLeNet$_{\text{[RGB+OFL]}}$ + NN	0.72	0.73	0.72	0.73
AlexNet$_{\text{[Fine-tuned]}}$	0.77	0.75	0.76	0.75
GoogLeNet$_{\text{[Fine-tuned]}}$	0.73	0.71	0.72	0.71
SqueezeNet$_{\text{[Fine-tuned]}}$	0.70	0.66	0.68	0.66
AlexNet$_{\text{[Fine-tuned]}}$ + SVM	0.76	0.72	0.74	0.72
GoogLeNet$_{\text{[Fine-tuned]}}$ + SVM	0.72	0.71	0.68	0.71
SqueezeNet$_{\text{[Fine-tuned]}}$ + SVM	0.64	0.59	0.61	0.59
3 CNNs + SVM	0.73	0.67	0.70	0.67
3 CNNs + NN	0.77	0.75	0.76	0.75
3 CNNs + NN + PP	0.77	0.76	0.75	0.76
3 CNNs + LSTM	0.78	0.78	0.78	0.78
3 CNNs + LSTM + PP	**0.80**	**0.78**	**0.79**	**0.79**

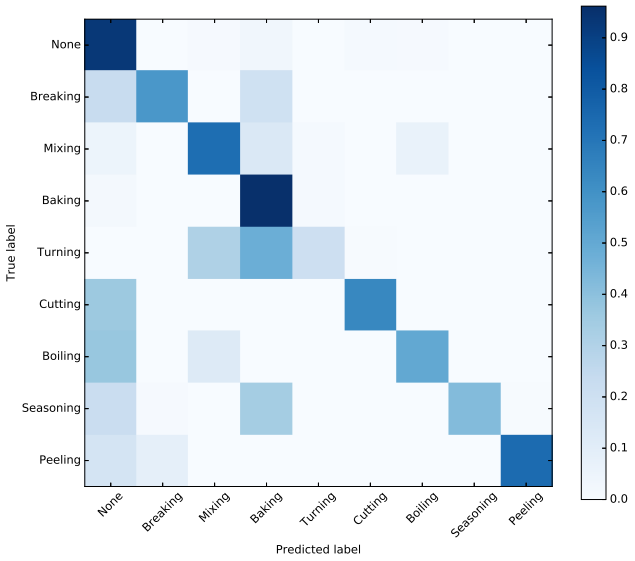

Fig. 5. Normalized confusion matrix for the *3CNNs+LSTM* network.

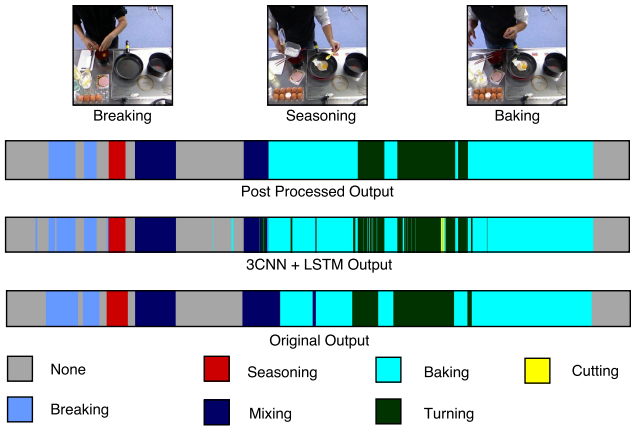

Fig. 6. Temporal representation of classes for the frame sequence of a single video in which true labels are compared with labels predicted by our approach (with and without post processing.

and columns the predicted classes. Shades of blue represent the value in each cell, going chromatically from a darker blue for higher values to a lighter blue for lower values. The confusion matrix shows normalized values, *i.e.*, predicted values are divided by the total number of true values for each cell. By analyzing the results for the *Baking* class, we can see that the system incorrectly predicts it as *Turning*. Such misclassification makes sense since both activities occur in the same region of the frame, using the same objects (*e.g.*, in both activities the subject is working in the middle of the scene and whereas in *Baking* the subject puts the broken egg onto the pan and lets it bake, in *Turning* the subject turns the baked egg on the pan. Even though both *None*, *Mixing*, and *Baking* have higher values in the main diagonal of the confusion matrix, their precision scores are reduced by the misclassification of other classes. Similarly to the misclassification of *Baking*, the *Mixing* and *Turning* activities occur in the same region of the scene and with the same objects.

Unlike *Baking* that does not have many changes through frames (*e.g.*, the egg baking inside the pan), the *None* activity is labeled as anything that happens but the other 8 activities, encompassing frames in which the subject is preparing the kitchen utensils, moving pans, and inter-activity frames such as removing the egg from boiling to peeling. The large accuracy (73%) for classifying *Baking* may be explained due to this standard behavior of low-variability in regions of the scene and the larger number of available training frames. Despite the unbalanced nature of the dataset, the values of accuracy follow the F-Measure scores, where the lowest value is achieved for *Turning* and the largest value for *Baking*.

7) Post processing effect: Since the post processing strategy seems to be successful, it is interesting to visualize the exact effect of smoothing predictions. Figure 6 presents the temporal representation of the class distribution in the frame sequence for a single video of the test set. Classes are represented by colored vertical lines in a temporal sequence for both the original output (true labels), the output provided by *3CNNs+LSTM*, and the output provided by (*3CNNs+LSTM+PP*). By analyzing the output of *3CNNs+LSTM*, we can see that some frames are misclassified such as a single *Mixing* action in the middle of the *Baking* class or a small sequence of the *Cutting* action in the middle of a *Seasoning* action. After performing the post-processing smoothing step, these noisy predictions disappear. On the other hand, small sequence of frames that were correctly classified (*Mixing* in the middle of a *Baking* class) also disappear. Despite the increase in accuracy (from 86% to 90% for the example presented in Figure 6), the smoothed output completely ignored the existence of some activities, which can be an important issue according to the application at hand.

IV. Related Work

Before the rise of CNN and neural networks in general, the approaches for action recognition were based on complex hand-crafted features extracted from video sequences [7]. Convolutional neural networks on the other hand, learn automatically a hierarchy of features automating the process of feature construction. Thus, many authors apply CNNs to recognize actions in videos using methods such as Long-term Recurrent Convolutional Network (LRCN) [16], 3D convolutions (3D CNNs) [18], or a mix of hand-crafted features and CNNs (two-stream CNNs) [15]. Despite the fact that these approaches are suitable for recognizing activities in general, they were applied in other datasets and are not directly comparable to our work.

Traditional approaches for activity recognition rely on hand-crafted features and domain-specific image processing algorithms and often result in limited accuracy [7], [19]. Bansal *et al.* [7] perform daily life cooking activity recognition based on hand-crafted features for hand movements and objects use in KSCGR dataset [6]. Their method first detects hand regions through color segmentation and skin identification. Since some objects can have the same color of the skin,

they perform background subtraction, eliminating still objects with skin color. Also, considering that objects may give hints of the activity (*e.g.*, the use of the knife may indicate the cutting activity), objects are identified as "*Not in use*" and "*In use*". A dynamic Support Vector Machine (SVM) and Hidden Markov Model (HMM) hybrid model combines the structural and temporal information to jointly infer the activity, achieving 64% of overall accuracy. In order to improve the performance of the system, they perform a post-processing step, removing noisy frames, *i.e.*, frames that are incorrectly classified among a cluster of correctly classified frames. Since some activities are temporally dependent of others, *e.g.*, *Peeling* only occurs after *Boiling*, they create a context grammar to select the the most likely guess for misclassified frames. Using the post processing step, Bansal *et al.* increased accuracy in $\approx 7\%$ for the activity recognition, achieving a final accuracy of 72%.

Ni *et al.* [20] propose an adaptive motion feature pooling scheme that utilizes human poses as side information. They extract hand-crafted features from the images, such as histogram of oriented gradient, motion boundary histogram, histogram of optical flow, and trajectory shape in order to obtain more relevant features. The principal component analysis (PCA) algorithm reduces the dimension of the large amount of extracted features. Improved Fisher vectors encode the resulting features and a second PCA algorithm reduces their dimensionality. Finally, they train a Linear SVM in order to classify video segments. They perform experiments using two datasets, the KSCGR dataset [6] and the MPII kitchen activity dataset [21]. Since their work have focused on object detection and tracking movements, they do not present results for activity recognition, not allowing us to make a fair comparison.

V. Conclusions and Future Work

In this work, we proposed a novel neural architecture for indoor fixed-camera kitchen activity recognition based on static and temporal data and different fusion methods. The pipeline of the architecture includes the training of deep convolutional neural networks to extract features from images and classify unseen frames. Using optical flow and RGB frames from the kitchen scene dataset (KSCGR), we performed experiments showing that the convolutional networks can indeed learn high-level relevant features for the activity recognition task at hand. Experiments show that our approach that employs fusion methods achieve better results when compared with the current state-of-the-art work that employs only hand-crafted features [7] or when compared with deep approaches that make use of RGB/OFL images alone. As future work, we intend to explore other approaches such as temporal pooling and employ other deep learning architectures such as 3D CNNs [18] considering that they are also capable of encoding temporal features in order to perform action recognition in videos.

Acknowledgement

This paper was achieved in cooperation with HP Brasil Indústria e Comércio de Equipamentos Eletrônicos LTDA. using incentives of Brazilian Informatics Law (Law nº 8.2.48 of 1991). The authors also would like to thank FAPERGS, CNPq, and CAPES for funding this research.

References

[1] G. Sukthankar, , C. Geib, , H. H. Bui, , D. V. Pynadath, , and R. P. Goldman, *Plan, Activity, and Intent Recognition*. Boston: Morgan Kaufmann, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780123985323000221

[2] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *CVPR*, 2014.

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

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[5] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

[6] A. Shimada, K. Kondo, D. Deguchi, G. Morin, and H. Stern, "Kitchen scene context based gesture recognition: A contest in icpr2012," in *Advances in depth image analysis and applications*. Springer, 2013, pp. 168–185.

[7] S. Bansal, S. Khandelwal, S. Gupta, and D. Goyal, "Kitchen activity recognition based on scene context," in *2013 IEEE International Conference on Image Processing*. IEEE, 2013, pp. 3461–3465.

[8] P. Sondhi, "Feature construction methods: a survey," *sifaka. cs. uiuc. edu*, vol. 69, pp. 70–71, 2009.

[9] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector machines," *Journal of machine learning research*, vol. 2, no. Dec, pp. 265–292, 2001.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[13] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *arXiv:1602.07360*, 2016.

[14] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Scandinavian conference on Image analysis*. Springer, 2003, pp. 363–370.

[15] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Advances in Neural Information Processing Systems*, 2014, pp. 568–576.

[16] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *CVPR*, 2015.

[17] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *The International Conference on Learning Representations (ICLR), San Diego*, 2015.

[18] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[19] L. Gorelick, M. Blank, E. Shechtman, M. Irani, and R. Basri, "Actions as space-time shapes," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 12, pp. 2247–2253, 2007.

[20] B. Ni, P. Moulin, and S. Yan, "Pose adaptive motion feature pooling for human action analysis," *International Journal of Computer Vision*, vol. 111, no. 2, pp. 229–248, 2015.

[21] 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 (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 1194–1201.

View publication stats