Who's Got the Groove?
Classifying Dance Style from Video

Noa Bendit-Shtull
Statistics Department
Stanford University
nbshtull@stanford.edu

Tiffany Cheng
Statistics Department
Stanford University
tiffc@stanford.edu

Julie Wang
Computer Science Department
Stanford University
juliesw@stanford.edu

Abstract

Many video action recognition models rely on background information or context clues; these models are natural extensions of image processing, where only spatial information is needed. Our project focuses on classification of dance styles from video, a domain that requires representation of motion because contextual information such as scenery or props is often sparse. We use Castro et al.'s Let's Dance dataset, designed specifically for this purpose. Our contribution is the application of sequential models to this dataset. We show that, while sequential models out-perform purely spatial models by roughly 10% accuracy, they are not as successful in dance style recognition as Castro's models that use optical flow to explicitly encode motion.

1. Introduction

Identifying highly dynamic, temporal actions from video is a complicated task in visual recognition. Of particular interest is identifying human motion and activity, with applications ranging from security, automatic video tagging, and collision avoidance systems. Humans display almost full range of motion through dance, and accurately identifying dance styles requires numerous samples over time. Thus, accurately identifying dance styles is an excellent case study for validating human motion identification methods.

Our project is an action recognition task which seeks to classify video clips as one of ten types (classes) of dance: ballet, break dancing, flamenco, foxtrot, latin, quickstep, square, swing, tango, or waltz. Some of these dances occur in very similar settings (e.g. the ballroom dances shown in Figure 1), which makes them challenging to differentiate based on background imagery alone. As a result, a successful model will need to encode movement or pose information.

For this task, we use the Let's Dance dataset presented by Castro *et al*. [4], which is comprised of 10-second video clips from the ten dance styles listed above. The input to our model is a sequence of RGB frames from a single video clip and the pose data extracted from those frames. The output of our model is a prediction of the type of dance in the video.

There are three broad classes of models used for action recognition in video: spatial, temporal, and sequential. A spatial model is a frame-by-frame model that only considers two-dimensional information. Spatial models are often used as the baseline methods. Temporal models incorporate a time dimension, for example by processing chunks of many frames rather than single frames, or by using 3D convolution. Sequential models, such as RNNs, are a subset of temporal models; they include a time dimension, but add the constraint that there is no leakage of information from the future to the past.

Castro *et al*. [4] present both spatial models and temporal models, but no sequential models. In this paper, we assess the effectiveness of several sequential models, including a traditional LSTM, in modeling dynamic movement. The most common challenge with recurrent models is the vanishing gradient problem. To address this problem, we also present an LSTM model with self-attention, and the more recent temporal convolutional network from Bai *et al*. [2], which does not suffer from the usual maladies of recursive networks. We find that, although these sequential models achieve significantly better performance than the baseline frame-by-frame approach, they don't achieve the accuracy of Castro's temporal models.

2. Related Work

Image recognition with convolutional neural networks has had great success in recent years, with state-of-the-art object detectors outperforming humans on the ILSVCR image recognition task [9]. However, action identification on videos has not reached the same level of technical maturity. Two major problems exist: first, videos are larger than im-

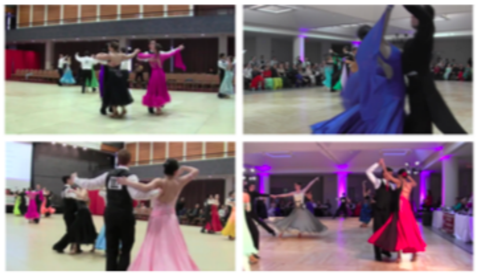


Figure 1: Example frames from the RGB dataset [4]. Class types from top left to bottom right: waltz, quickstep, foxtrot, tango.

ages, so training is much more computationally expensive. Second, to perform well, video classifiers must reason about the relationships between features over space and time. This is harder than the image classification task, which only has to learn spatial relationships. A large body of prior work exists to address these issues.

Early video classification methods use hand-crafted features and non-neural network models to classify videos. Custom features include 'Histogram of Gradient' and 'Histogram of Optical Flow' [14], as well as the popular 'dense trajectory' features from Wang and Schmid (2012) [28]. These handcrafted features are not generalizable, and cannot be used on a wide variety of videos. Early models include bag of words [19] or Fisher vectors [22]. These models are typically less costly to train and evaluate, but lack the expressivity and capacity of neural network models. While these approaches mostly been supplanted by deep learning methods, some handcrafted features, notably optical flow, are still used as pre-extracted features in neural network models.

When convolutional neural networks started gaining popularity, video classification also shifted to deep learning methods. Early works treat spatial and temporal domains of an image equally, performing 3D convolution on stacks of image frames. Ji *et al.* [10] perform 3D convolutions on inputs consisting of multiple stacked continuous frames. In order to capture longer term dependencies, they hand-craft auxiliary long-term features. Further work by Karpathy *et al.* [11] builds on 3D convolution by introducing different ways to fuse the information from a stack of frames. They treat each video as a bag of short clips. Each clip is passed through a CNN, and the results are combined with early, late, or slow fusion in an attempt to capture temporal features across the clip. In early fusion, the first layer of the network performs a 3D convolution on multiple input frames. In late fusion, temporally distant frames are passed to separate convolutional stacks, which are trained in parallel, and the results are fused in a final linear layer. Slow fusion is a mixture of these two approaches, and works marginally better than either. Despite these techniques, Karpathy *et al.* [11] only achieve marginally better performance than single frame models, which don't attempt to learn any temporal relationship between images.

This failure to learn good temporal features was addressed by Ng *et al.* [30], who applied temporal pooling [29] to convolutional neural networks. Temporal pooling computes features for every frame, then pools them across frames. The pooled result is used to classify the video. Ng *et al.* [30] find that max pooling over the outputs of the last convolutional layer in a network yields better results than early, late, or slow fusion.

Another key development in capturing temporal video features is the use of optical flow data. Optical flow data captures the motion of each pixel between the current frame and subsequent frames. Typically, Farneback's methodology [7] is used to calculate the pixel displacement. An immensely influential paper, [23], finds that optical flow data is critical to making accurate video classifiers, and is able to achieve 81.2% classification accuracy on UCF-101 using both RGB and optical flow information, the best at the time. This need for optical flow data is echoed in most later works, including [8] and [15]. Two-stream CNNs using RGB and optical flow remain one of the most popular and effective approaches for action recognition.

All of the techniques mentioned thus far make use of convolutional neural networks with a final linear layer. The major drawback of this technique is that long term dependencies cannot be captured. Other works seek redress with the use of alternative architectures, including Recurrent Neural Networks. [30] was one of the first to investigate the use of an LSTM-RNN, but they find that it underperforms temporal pooling of convolutional features. Ma *et al.* (2017) [15] develop a Temporal Segment LSTM; in a TS-LSTM, the sampled video frames are divided into segments, and a temporal pooling layer is applied to extract the best features from each segment. Those extracted features are then fed into a single layer LSTM. They find that the temporal segmentation of the videos is critical, as is the use of a single LSTM layer to avoid over-fitting. Using these methods, they are able to achieve 94.1% accuracy on the UCF-101 dataset. To our knowledge, this is the best performing RNN architecture on the UCF-101 classification task.

To compare the performance of these various algorithms on action classification tasks, a summary of their performance on the UCF-101 dataset [24] is included in Table 1. [1].

There is limited available research on identifying dance styles from video. The research that does exist centers around cultural folk dances, and focuses on methods for pose estimation. Protopapadakis *et al.* [20] focus on using pose information extracted from sensors to identify six

Methods	UCF-101
UCF-101 Baseline [24]	43.9%
3D CNN with Slow Fusion [11]	63.3%
Two-Stream [23]	88.0%
Convolutional Two-Stream [8]	92.5%
TS-LSTM [15]	94.1%

Table 1: Summary of video classification model accuracy on UCF-101 Dataset.

types of Greek folk dance. The authors also encode temporal information by subtracting the pose information in consecutive frames. Similarly, Kishore *et al*. [13] focus on classifying Indian classical dance (ICD) actions using CNNs, but the focus is on identifying dance forms through dance poses as opposed to distinctive movements. For example, one way that the authors generate their training data is recording different subjects performing 200 familiar dance poses from various ICD forms, making sure to capture each pose for at least 60 frames. Dewan *et al*. [6] use pose estimation to extract RGB and optical flow data for regions of "important motion" in Indian classical dance. They use a CNN to extract features from each region, and combine the features for each frame. Finally, they use an RNN to model a sequence of frames, achieving 93.6% classification accuracy on the ICD dataset.

Our work focuses on the use of RNN architectures, as they are much more capable than 3D convolutions at capturing long term time dependencies. We make use of a late fusion, two-stream architecture, as this approach has been shown to yield excellent, computationally tractable results. Finally, we erroneously excluded optical flow from our work, as we did not realize at the outset how important that additional temporal encoding was to classification accuracy.

3. Methods

3.1. Baseline Methods

The traditional baseline method for action classification from video is a spatial model—that is, a frame-by-frame model with no temporal component. A CNN is trained on individual frames of a video. At test time, all frames of a video are individually classified, and the video's final classification is the majority class of its frames. We compare our results to the frame by frame model introduced by Castro *et al*., which uses a pre-trained variant of CaffeNet that has been finetuned on the Let's Dance dataset. This model achieves test accuracy of 56.4% [4].

Additionally, we compare our model to the five temporal models presented by Castro *et al*., summarized in Table 2. These models incorporate the time dimension in one of three ways: optical flow data, 3D convolution, or stacked convolution. The optical flow data is computed from frame n and frame $n - k$, using Farneback's methodology [7]. 3D convolution uses a receptive field that spans both space and time dimensions. Stacked convolution uses 16 frames are stacked together to form an input; this is to capture temporal dependencies between the frames.

The most successful of Castro *et al*.'s models are the temporal 3D CNN using RGB data only and the temporal three-stream CNN. The drawback of the 3D convolution is that it is computationally intensive to the extent that it was not possible to train a three-stream model with 3D convolution. The temporal three-stream CNN has similar performance (70.11% vs 71.6% for the 3D convolution), but is less computationally intensive. However, Castro *et al*. note that the single-frame two-stream and three-stream CNNs achieve almost the same accuracy as the temporal three-stream CNN without stacked convolution. What these three models have in common is the use of optical flow data; which raises the question of whether optical flow is the key reason these multi-stream models were successful.

3.2. Proposed Approach

To answer this question, we propose three models that do not use optical flow data to capture the temporal dimension. Instead, we rely only on spatial data from each frame, and use sequential models to capture motion over time.

Like Castro *et al*., our method utilizes a two-stream late fusion model, shown in Figure 2. The two streams of data are RGB images and PoseNet skeleton data. Features are extracted from each RGB frame using Pytorch's [18] pre-trained ResNet-18 model with the final layer converted to an identity layer, resulting in 512-dimensional features.

Features are extracted from the PoseNet data using a shallow CNN with small filters adapted from Khalid and Yu [12]. Specifically, the model has two 2D convolutions with three channels and kernel size three, a ReLU activation, and a fully connected layer with input size 102 and output size ten (the number of classes). The model weights are initialized using Xavier initialization, and the model is trained with an Nesterov accelerated gradient descent (momentum 0.9), batch size 100, and learning rate 0.01. Note that while Castro *et al*. train their model on the "visualized" pose data — RGB images of the extracted pose skeleton

Method	Data Streams	# Frames	Convolution	Testing Accuracy
Frame-by-Frame CNN	RGB	1	2D	56.4%
Two-Stream CNN	RGB; Optical Flow	1	2D	68.89%
Three-Stream CNN	RGB; Optical Flow; Pose	1	2D	69.20%
Temporal 3D CNN RGB	RGB	16	3D	70.11%
Temporal 3D CNN Skeletal	Pose	16	3D	57.14%
Temporal Three-Stream CNN	RGB; Optical Flow; Pose	16	2D	71.6%

Table 2: Summary of models presented by Castro *et al.* [4]. The frame-by-frame CNN is a spatial model, while the remaining five models are temporal, incorporating the time dimension using optical flow data, 3D convolution, and/or stacked convolution using 16-frame chunks.

— we choose to train our model on the raw feature maps. These features are much richer; they exactly capture positions of joints and limbs. Training a CNN on the visualized skeletons reintroduces a layer of abstraction.

Note that both feature extractors are trained on individual frames so no temporal aspects are considered at the feature extraction level.

After extracting the features from both datasets, we concatenate the 512-dimensional RGB image features and the 102-dimensional pose features to create 614-dimensional representations of each image. To create sequences from these features, we subsample the original 30fps video clips, with the subsampling rate determined through hyperparameter tuning. This creates a sequence of length anywhere between 2 and 300 for each 10-second clip. This sequence of combined data is then input to one of three sequential models: an LSTM, an LSTM with self-attention, and a temporal convolutional neural network (TCN).

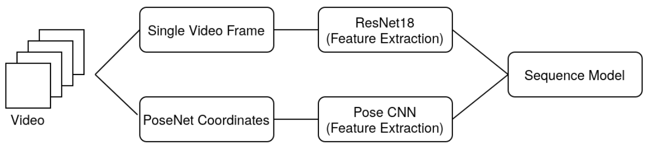


Figure 2: Our Two-Stream Late Fusion Architecture using raw input frames and PosetNet coordinates. The sequential model is either a vanilla LSTM, LSTM with self-attention, or temporal convolutional neural network (TCN).

3.2.1 LSTM

This network consists of a dropout layer, a LSTM with hidden dimension 100, and a fully-connected layer to produce scores for the ten classes. The hyper-parameters tuned are the learning rate, dimension of the hidden layer, batch size, dropout rate, weight decay (equivalent to L2 regularization), frame sampling frequency, and optimizer (Adam or SGD). Our final model is trained with learning rate 0.0167, hidden dimension 128, batch size 64, dropout rate of 0.7, weight decay 0.0284, frame sub-sampling of every 60 frames, and Nestorov accelerated gradient descent (momentum 0.9).

3.2.2 LSTM with Self-Attention

In our LSTM, the addition of self-attention allows the model to determine which frames in a sequence should receive the most weight in the output. The attention mechanism used is scaled dot product self-attention, adapted from Vaswani *et al.* [27]. Vaswani *et al.* advocate for using self-attention as opposed to the recurrent and convolutional layers commonly used for attention because it is computationally less complex and faster to compute. Self-attention also yields more interpretable models.

In self-attention, the weights are determined by computing the dot product of the final cell state, c_T, and the outputs from each step in the sequence, $y = [y_1, y_2, \ldots, y_T]$. These weights are then divided by the dimension of the outputs and constrained with a softmax. The result α is used to compute a linear combination of the outputs y_i as seen in the equation below:

$$a = \left[\frac{c_T^T y_1}{\sqrt{d_y}}, \frac{c_T^T y_2}{\sqrt{d_y}}, \ldots, \frac{c_T^T y_T}{\sqrt{d_y}} \right], \quad \alpha = \sigma(a)$$

Figure 3 visualizes the computations necessary for a LSTM with scaled dot-product self-attention.

For the self-attention model, we tuned the same hyper-parameters as for the plain LSTM model. The resulting model is trained with learning rate 0.0591, hidden dimension 32, batch size 16, dropout rate of 0.05, weight decay 1e-05, frame sub-sampling frequency of every 30 frames, and Nestorov accelerated gradient descent (momentum 0.9).

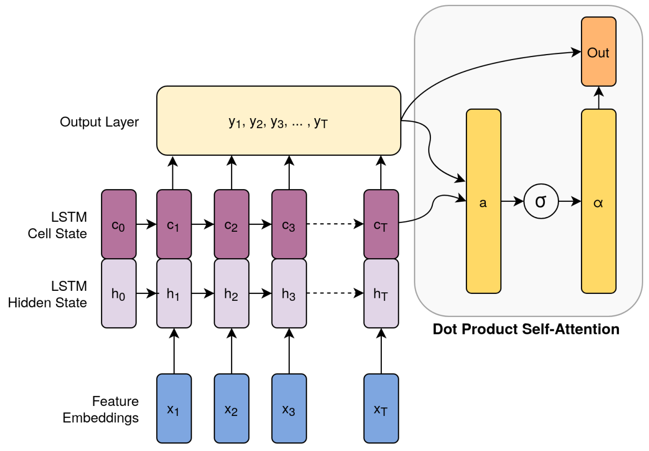


Figure 3: LSTM layer with self-attention.

3.2.3 Temporal Convolutional Network

We use a temporal convolutional network (TCN) introduced by Bai *et al.* (2018) [2, 5]. The authors describe a TCN as a network that (1) is *causal*, meaning that the receptive field of each convolution only includes previous elements in the sequence, and (2) can process a sequence of any length. A TCN is similar to a RNN in that it processes inputs sequentially, and it can process a sequence of any length. However, it does not suffer from the vanishing gradient problem on long sequences. This is because the direction of backpropagation is not through previous elements of the sequence, but through the previous convolutional layers. This feature makes the TCN a compelling alternative to our LSTM architectures.

Each layer of the TCN architecture contains causal convolutions with increasing dilation. A causal convolution is a convolution that, at time t, only sees elements of the previous layer at time t or an earlier time [2]. A convolution with dilation is one with gaps between rows and columns of the filter. In the one-dimensional case, a causal dilated convolution F with kernel size k can be computed on input x at time t as:

$$F(t) = \sum_{i=1}^{k} f(i) \cdot x_{t-d\cdot i}$$

where d is the dilation factor. Dilation is used to create a receptive field that grows exponentially in each subsequent layer. The effective history of each layer in a TCN is $(k - 1) \cdot d$.

Figure 4 shows a three-layer TCN with kernel size 3 and exponential dilation. The sequences are padded to ensure that each hidden layer has the same sequence length. The output layer in this diagram has effective history of $(3 - 1)4 = 8$. The receptive field is even larger; if the elements of the nth layer have effective history $(k - 1)2^{(n-1)}$, then the receptive field of the nth layer is

$$\sum_{i=1}^{n} (k - 1)2^{(n-1)}$$

The TCN used in this study also has kernel size 3 and exponential dilation, but it has six layers. The dilation factor in the sixth layer is 2^5, so the effective history is 64 and the receptive field is 126.

The hyperparameters tuned were the batch size, number of layers, number of convolution channels, dropout probability, learning rate, and optimizer. The final model was trained with batch size 64, 6 layers, 128 channels per convolution, dropout rate of 0.02, learning rate 0.0022, and the same SGD optimizer used for the LSTM models. The convolutional filters for each layer were randomly initialized using a $\mathcal{N}(0, 0.01)$ distribution. The frame sub-sampling is every five frames.

4. Data

We use the publicly available Let's Dance video dataset [3] created by Castro *et al.* [4], which contains roughly 1,000 videos spanning 10 dynamic and visually overlapping dance types. There are approximately 100 videos per dance style, extracted from YouTube in 10-second clips at 30 frames per second. Each video is available in its original RGB format and as extracted pose coordinates.

The data was randomized at the video level into 80% training, 10% validation, and 10% test data. The same splits were used for the RGB data and the pose data.

4.1. RGB Data

Each frame in the RGB dataset is stored as a JPG file (see Figure 1 for examples). We process the frames using a series of Pytorch image transformations. First, the images are resized to (3x256x256), then they are normalized using the mean and standard deviation from ImageNet as recommended by PyTorch's documentation for pre-trained models [21].

4.2. Pose Data

Each frame's pose data is a set of (x, y) coordinates for 17 different body labels for each person in the frame. Body labels include nose, left eye, and right hip. The number of dancers per frame varies between zero and 20. An empty pose file typically indicates that the feature extractor was unable to identify any bodies, while a pose file with 20 entries indicates a group performance. Although our data source states that the pose data was extracted using Facebook's DensePose, the data does not match the DensePose schema. Based on the 17 body labels in the pose data,

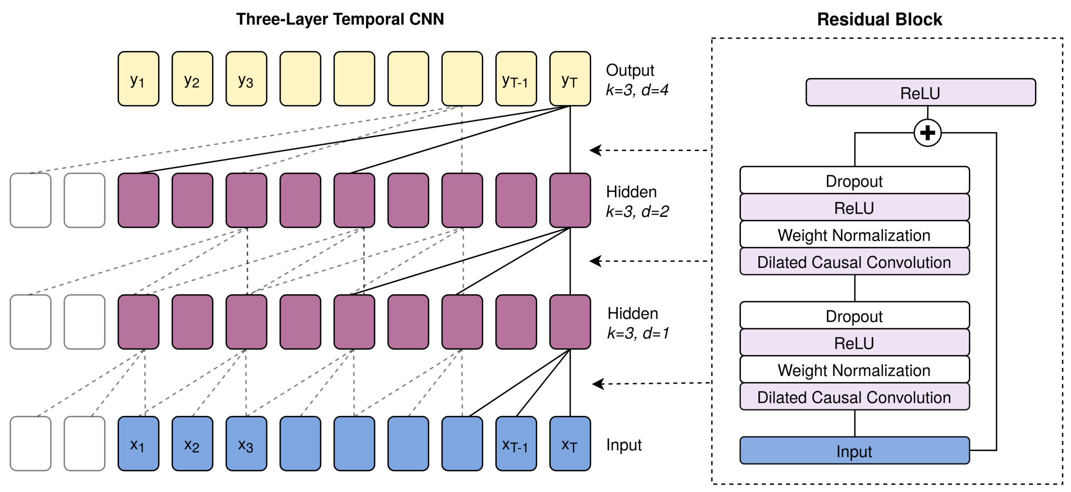


Figure 4: Architecture of a TCN. The left-hand side shows a three-layer TCN with kernel size 3 and exponential dilation in each layer. The right-hand side shows the detail of each layer, including two dilated causal convolutions and a residual connection. Based on Figure 1 from [5]

it seems that the poses were created using TensorFlow's PoseNet model [26].

To transform the pose data into a usable format for our models, we created a one time pre-processing pipeline. We zero-pad frames with fewer than 20 skeletons and generate missing pose files for 28 missing frames. Next, we center and normalize the "keypoints" of each skeleton to achieve a standardized dancer. This is necessary for several reasons: there are varying frame sizes in the dataset, dancers appear in different parts of the image, and dancer scale varies based on camera proximity. To center and normalize all the keypoints, we use a similar approach as Neverova *et al.* [16]-we construct a bounding box around each skeleton and use that box to normalize the keypoints. This results in keypoint values between -1 and 1.

5. Results

Our quantitative results, relative to the frame-by-frame baseline from Castro *et al.*, are shown in Table 3. Although the frame-by-frame model is the most relevant, it is not a perfect comparison. Castro's frame-by-frame model makes predictions for every frame in a given video clip, and then uses plurality voting to classify the video as a whole. In contrast, our sequential models use frame sub-sampling to reduce the length of the input sequence, so less information is used for training. Furthermore, all of our models utilize the extracted pose information, which Castro's frame-by-frame baseline does not.

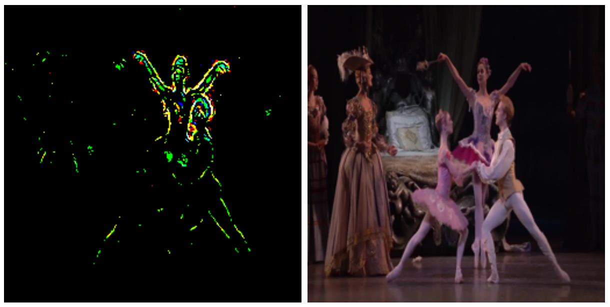
We want to note that the differences in testing accuracy presented in Table 3 may stem from more than just variations in modeling technique. On the Let's Dance project site [3], where we obtained our dataset, Castro *et al.* [4] note that some of the videos used in their model had been removed from the dataset since they had been deleted from Youtube. Furthermore, since publishing their paper, Castro *et al.* [4] have changed the way they generate pose data from the raw video frames.

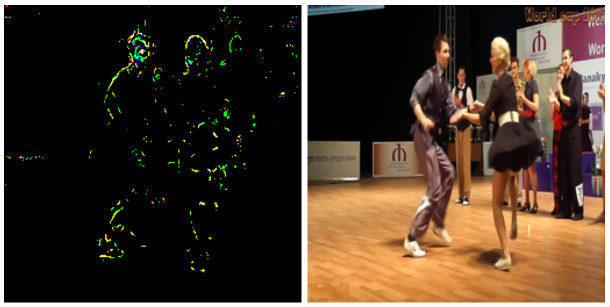
5.1. Feature Encoding

In order to assess what information the pre-trained ResNet-18 encoded from the RGB images, we created saliency maps to visualize the 512 features in the output layer. In particular, we visualized the gradients of the features with respect to the input images using the "guided backpropagation" method introduced by Springenberg *et al.* [25]. We created saliency maps for a sample of video frames, and for each frame visualized the average saliency map across 512 features, as shown. This average was computed by summing the 512 saliency maps and dividing the result by 512. Figure 5 shows samples from three dance styles. The saliency maps demonstrate that the features extracted from the RGB images are generally focused on pose information rather than the scenery. We observed that the features consistently picked up on the people's bodies (particularly their heads), as well as light fixtures and prominent edges in the background. We also noticed that the ResNet was confused by fabric, and failed to pick up people wearing long dresses or skirts, as in Figure 5a.

5.2. LSTM

The LSTM model achieves testing accuracy of 65.9%, ten percentage points higher than the baseline frame-by-frame model. However, the accuracy is still several per-


(a) Ballet


(b) Swing

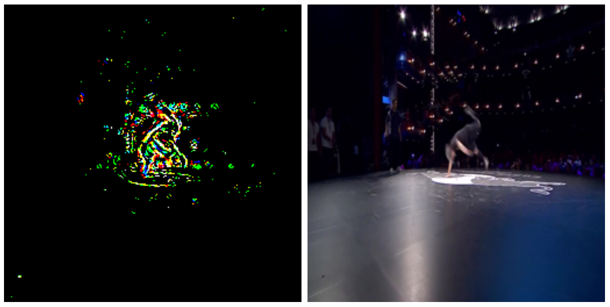

(c) Break Dance

Figure 5: Saliency maps of RGB images computed using guided backpropagation through the pretrained ResNet-18 architecture.

centage points behind Castro's multi-stream CNNs that incorporate optical flow.

The dropout rate used to train this model is 0.70; because the data set is so small (less than 1,000 video clips in total), even a small model can be expressive enough to over-fit the training data. Therefore, aggressive dropout is needed to ensure that the model generalizes to the test data.

Surprisingly, the hyperparameter tuning chose subsampling of every 60 frames, meaning that each 10-second video clip was reduced to a sequence of 5 frames, one every two seconds. This level of sub-sampling is surprising because it can't encode dance movements faster than two seconds. On the other hand, it's likely that processing very short sequences greatly reduced or eliminated the vanishing gradient problem.

Figure 6a shows the confusion matrix for the LSTM model. The dance styles predicted with the greatest accuracy are ballet, break dancing, and square dancing. Two videos of Latin dance were classified as ballet, and one video of break dancing was classified as swing. Ballet and break dancing, unlike the other dance forms, are not partner dances, so they are visually distinctive. Although square dancing is a partner dance, large portions of the dance are performed in groups, for example by joining hands in a circle. This may explain why square dancing is always correctly identified, and is never confused with other dance forms. The highlighted area in the bottom right corner of the confusion matrix indicates that tango and waltz are often confused; of the ten tango videos, half were mistakenly classified as waltz. Quickstep is the most frequently misclassified; only two of the ten videos are correctly identified as quickstep, while the remainder are marked as foxtrot, swing, tango, and waltz.

5.3. LSTM with Self-Attention

We expected self-attention to enhance the performance of the LSTM by directing weight to the most important frames. However, the LSTM with attention achieves accuracy of 63.7%, slightly lower than the standard LSTM. Given that our test data set only consists of 91 video clips (10% of the original 1000 videos), this performance is comparable. The confusion matrix in Figure 6b is similar to the confusion matrix for the LSTM. The LSTM with attention correctly classifies all 11 ballet videos, 10 break dancing videos, and 9 square dance videos, but does not classify *any* foxtrot videos correctly. Relative to the plain LSTM, it is more successful at differentiating tango and waltz; perhaps the attention mechanism was useful in differentiating visually similar ballroom dances. There were no dance types where the TCN outperformed both LSTM models in identifying the right videos.

Even though classification accuracy was comparable between the LSTM model and the LSTM with attention model, hyperparameter tuning yielded pretty different parameters. While hyperparameter tuning chose a very aggressive dropout rate for the LSTM, it selected a dropout rate of 0.05 for the self-attention model. Also, it preferred a smaller frame subsampling rate of every 30 frames for the self-attention model, meaning that each 10-second video clip was reduced to a sequence of 10 frames (one per second). This yields a sequence with double the number of frames as compared to the LSTM. Perhaps a smaller sampling rate was preferred because the attention model is able to handle longer sequences of frames since it can upweight more important frames and downweigh less important ones.

Method	Testing Accuracy
Frame-by-Frame CNN [4]	56.4%
Two-Stream LSTM	65.9%
Two-Stream Attention LSTM	63.7%
Two-Stream TCN	58.2%

Table 3: Comparison of testing accuracy against Castro *et al*. [4] frame-by-frame CNN.

5.4. Temporal Convolutional Network (TCN)

The TCN model achieves accuracy of 58.2% on the test data, only marginally better than the frame-by-frame model. Although the TCN performed well on the validation set, achieving 75% accuracy, it did not generalize well to the test data. The confusion matrix shows that, like the LSTM models, the TCN performs well on ballet, break dancing, and square dancing, likely due to the distinctive features of those dances. However, it struggled more than the LSTM models in identifying the tango videos, instead mislabelling most of them as waltz.

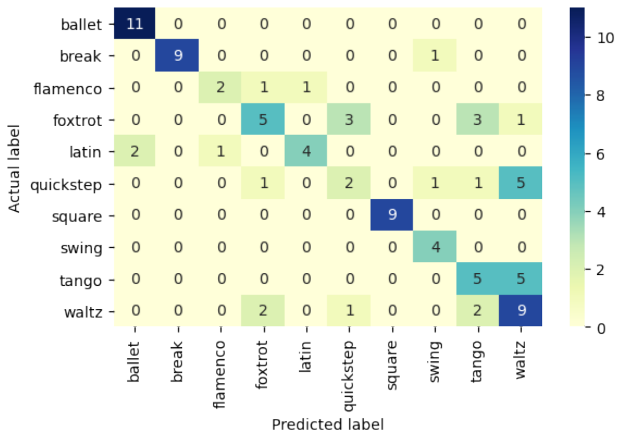
One hypothesis for the TCN's subpar performance is that it is not expressive enough to capture differences in movement between the dances; the LSTM models involve far more parameters per layer than the TCN. Perhaps if we had increased the complexity of the TCN by adding hidden layers, reducing the regularization, etc. its performance would have improved.

6. Conclusion and Future Work

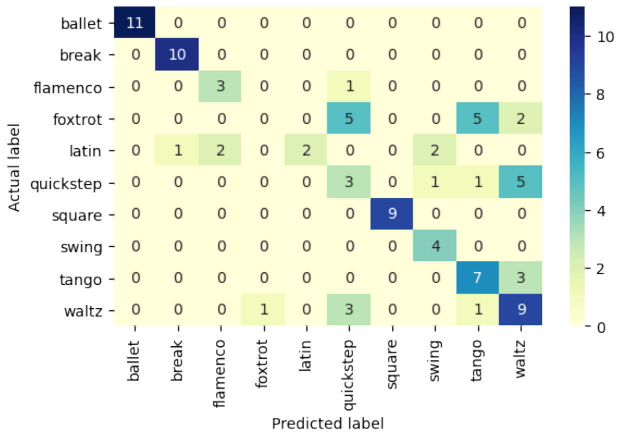
Surprisingly, the plain LSTM model out-performed both the LSTM with attention and the TCN model. We had expected that attention would help direct the focus to key frames, and that the TCN would alleviate potential problems with vanishing gradients. However, given that the LSTM performed best with only 5 frames from a 10-second video clip, vanishing gradients were not problematic.

Future work should expand hyperparameter tuning; we were able to tests hundreds of hyperparameter combinations, but Castro *et al*., for example, used 10,000 iterations to fine-tune their frame-by-frame baseline. Given the low accuracy of the TCN, we would like to explore deeper models, larger kernel sizes for the causal convolution, and different dilation strategies. We would also have liked to explore the TS-LSTM architecture, which selects the best out of every n frames for the RNN, rather than our method of using every nth frame.

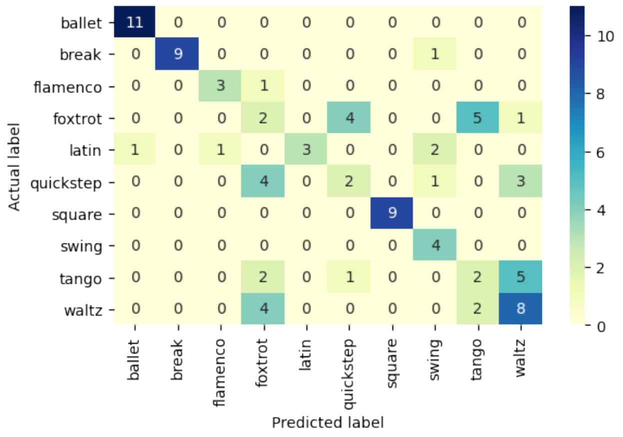
For this project, we chose not to use optical flow to encode the temporal dimension in order to determine how much a sequential model could capture on its own. However, we realized that optical flow is an important driver in the success of action recognition on highly dynamic motion. Given more time, we would set up a pipeline to process the videos' optical flow data and expand our models to be three-stream.



(a) LSTM



(b) LSTM with Self-Attention



(c) TCN

Figure 6: Confusion matrices showing model results.

Ultimately, one of the biggest constraints was the limited data set. Since there were only around 1,000 videos in total, the validation set and test set each only contained 100 observations. Castro *et al.* circumvented this problem by processing videos in 16-frame chunks, but we opted to use each video clip as a single input to our sequential models. Therefore, an important direction for future work is to use data augmentation to increase the data set size. This could include strategies such as blurring, jittering, shifting, or rotating videos. Future work could also accommodate this smaller dataset by exploring alternative regularization strategies. We employed dropout for all three models, weight decay for the LSTM models, and weight normalization for the TCN. However, normalization layers might also be helpful for the LSTM models.

7. Contributions and Acknowledgements

We would like to thank De-An Young and Ranjay Krishna, members of the CS231n course staff, for providing guidance and project ideas.

Within our team, Tiffany pre-processed the PoseNet data, built and tuned the LSTM model with attention, and created confusion matrices to visualize and compute model performance. Julie built the models for the pose data and the baseline LSTM model, tuned the TCN model, created saliency maps, and conducted the literature review. Noa designed the data pipeline for the raw image and processed pose data, set up the TCN model, analyzed the impact of frame sub-sampling frequency, and created visualizations of model architecture. All contributed to writing infrastructure code, debugging, and writing the reports.

The TCN model from Bai *et al.* was taken from `https://github.com/locuslab/TCN` [2, 5]. The saliency maps with guided backpropagation were adapated from [17].

References

[1] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. *CoRR*, abs/1609.08675, 2016. 2

[2] S. Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv:1803.01271*, 2018. 1, 5, 9

[3] D. Castro, S. Hickson, P. Sangkloy, B. Mittal, S. Dai, J. Hays, and I. Essa. Let's dance dataset, 2018. 5, 6

[4] D. Castro, S. Hickson, P. Sangkloy, B. Mittal, S. Dai, J. Hays, and I. Essa. Let's dance: Learning from online dance videos. *arXiv preprint*, 1801.07388, 2018. 1, 2, 3, 4, 5, 6, 8

[5] CMU Locus Lab. Sequence modeling benchmarks and temporal convolutional networks (tcn), 2018. [Online; accessed June 5 2020]. 5, 6, 9

[6] S. Dewan, S. Agarwal, and N. Singh. A deep learning pipeline for indian dance style classification. page 7, 04 2018. 3

[7] G. Farnebäck. Two-frame motion estimation based on polynomial expansion. volume 2749, pages 363–370, 06 2003. 2, 3

[8] C. Feichtenhofer, A. Pinz, and A. Zisserman. Convolutional two-stream network fusion for video action recognition. *CoRR*, abs/1604.06573, 2016. 2, 3

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 1

[10] 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*, 35(1):221–231, 2013. 2

[11] 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. 2, 3

[12] M. Khalid and J. Yu. Multi-modal three-stream network for action recognition. *IEEE ICPR*, 2018. 3

[13] P. V. V. Kishore, K. V. V. Kumar, E. K. Kumar, A. S. C. S. Sastry, M. T. Kiran, D. A. Kumar, and M. V. D. Prasad. Indian classical dance action identification and classification with convolutional neural networks. *Advances in Multimedia*, 2018, 01 2018. 3

[14] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008. 2

[15] C. Ma, M. Chen, Z. Kira, and G. AlRegib. TS-LSTM and temporal-inception: Exploiting spatiotemporal dynamics for activity recognition. *CoRR*, abs/1703.10667, 2017. 2, 3

[16] N. Neverova, J. Thewlis, R. A. Güler, I. Kokkinos, and A. Vedaldi. Slim densepose: Thrifty learning from sparse annotations and motion cues. 2019. 6

[17] U. Ozbulak. Pytorch cnn visualizations. `https://github.com/utkuozbulak/pytorch-cnn-visualizations`, 2019. 9

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 3

[19] X. Peng, L. Wang, X. Wang, and Y. Qiao. Bag of visual words and fusion methods for action recognition: Comprehensive study and good practice. *CoRR*, abs/1405.4506, 2014. 2

[20] E. Protopapadakis, A. Voulodimos, A. Doulamis, S. Camarinopoulos, N. Doulamis, and G. Miaoulis. Dance pose identification from motion capture data: A comparison of classifiers. *Technologies*, 6, 03 2018. 2

[21] PyTorch. Torchvision.models, 2020. [Online; accessed May 13 2020]. 5

[22] J. Sanchez, F. Perronnin, T. Mensink, and J. Verbeek. Image Classification with the Fisher Vector: Theory and Practice. *International Journal of Computer Vision*, 105(3):222–245, Dec. 2013. 2

[23] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 568–576. Curran Associates, Inc., 2014. 2, 3

[24] K. Soomro, A. R. Zamir, and M. Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402, 2012. 2, 3

[25] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. 04 2015. 6

[26] Tensorflow. Pose detection in the browser: Posenet model, 2020. [Online; accessed April 25 2020]. 6

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, and I. Polosukhin. Attention is all you need. *Conference on Neural Information Processing Systems*, 2017. 4

[28] H. Wang and C. Schmid. Action recognition with improved trajectories. In *2013 IEEE International Conference on Computer Vision*, pages 3551–3558, 2013. 2

[29] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, and C. Schmid. Evaluation of local spatio-temporal features for action recognition. In A. Cavallaro, S. Prince, and D. Alexander, editors, *BMVC 2009 - British Machine Vision Conference*, pages 124.1–124.11, London, United Kingdom, Sept. 2009. BMVA Press. 2

[30] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 2

We also used the following Python packages: *pytorch, scikit-learn, pandas, numpy, test_tube, matplotlib, PIL, tqdm*.