Identifying Player Actions in Tennis from Visual Data

Harsh Malara
College of Information and Computer Science,
University of Massachusetts, Amherst.
hmalara@umass.edu

Shivam Utreja
College of Information and Computer Science,
University of Massachusetts, Amherst.
sutreja@umass.edu

Abstract

Sports is a field with a huge abundance of high quality visual data. This data holds immense potential to be utilized with the state of the art techniques from computer vision and artificial intelligence, for a variety of tasks. One such task is of action recognition. In the general action recognition task in computer vision, each individual action is highly distinct and the action categories are coarse-grained. However, identifying different actions within a single sport requires a much more fine-tuned action categorization. This project's main goal was to make an open source implementation of one such model [8], which recognizes fine-grained action categories in tennis, given the video input of the player. This was followed by experiments with the architecture choice, feature choice and finally, comparison of sequential processing of an LSTM with the time-independent processing, by a pre-trained CNN.

1. Introduction

Identification of actions using visual data can be really useful for amatuer players and coaches who can use the video captured during playing to analyse mistakes and devise new strategies. For general application of the model we did not train the feature extractor for a specific dataset, but tried various general feature extractors trained on some another huge dataset. We also made sure that the videos used at train time were normal RGB videos (as opposed to silhouettes, depth, 2D skeleton etc.). We wish to use the power of these general purpose, pre-trained feature extractors in a more specialized setting, where the task is restricted to identifying tennis strokes from video.

These general features extracted for all the frames of the video as a whole, have information to predict the final action taken by the player. In order to incorporate this temporal dependency understanding in a model, we used the LSTM architecture for a sequence to label mapping which is trained from scratch. The input to this LSTM is the features extracted from all the frames of a video, in sequence; and the LSTM outputs a single label as output, which is the action being performed in the video.

2. Background

This sections provides a brief overview of the key ideas needed in order to understand the approach taken for the action recognition task.

2.1. Convolutional neural networks

CNNs or convolutional neural networks are a type of deep learning model architecture which has proven to work especially well on image classification tasks, by focusing on learning locally constrained spatial features specific to the task at hand. In the recent years, several architectures have been trained on the image classification task on very large datasets like Imagenet[2], that has 14 million images belonging to thousands of categories. The ResNet[4] and the Inception network [9] are two such networks that work particularly well on this task. It has also been shown that the features learnt in the deeper layers of such architectures are general purpose descriptions of the various objects occurring in the image. This general purpose nature of the features allows us to use the model trained on one dataset (say, the ImageNet) directly on other datasets to extract features which can be processed further for the other task. This process of using pre-trained feature extractor from one task, on another closely related task is known as transfer learning.

2.2. Recurrent networks and LSTM

Recurrent neural network is a type of deep learning model which is particularly useful in cases where the sequential nature of the information at the input or output (or both) needs to be exploited in order to perform well at the task. Examples of such tasks are image captioning (single input, sequential output), sentiment classification (sequential input, single output), machine translation (sequential input, sequential output) etc. The update equations for a single hidden layer vanilla RNN are shown below. The hidden state is updated in a recurrent fashion which helps it capture the sequential nature of the information.

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t) \tag{1}$$

$$y_t = W_{hy}h_t \tag{2}$$

Here, x_t, y_t, h_t are the input, output and hidden state respectively at the t^{th} time step.

However, vanilla RNNs face the problem of vanishing gradients. This means that the hidden state value is unable to retain the information from several states in the past. This is a major drawback, because most sequential prediction tasks can benefit if the model can selectively retain the key information from several time steps in the past.

This is where LSTMs[5] come into the picture. LSTM is a recurrent architecture which solves the vanishing gradients problem of the vanilla RNN. It introduces a separate 'cell state' for every hidden node in the network, from which previous information is selectively removed and new information is selectively added at every time step, with the help of 'gates' computed at every time step. The update equations of the LSTM are shown below;

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{3}$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{4}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{5}$$

$$\tilde{C}_t = tanh(W_C[h_{t-1}, x_t] + b_C) \tag{6}$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t \tag{7}$$

$$h_t = o_t \circ tanh(C_t) \tag{8}$$

Here, f,i and o are the forget, input and output gate respectively, which are also learnt during training. C_t is the cell state at time t, and h_t is the hidden state (i.e., the output of the hidden node) at time t.

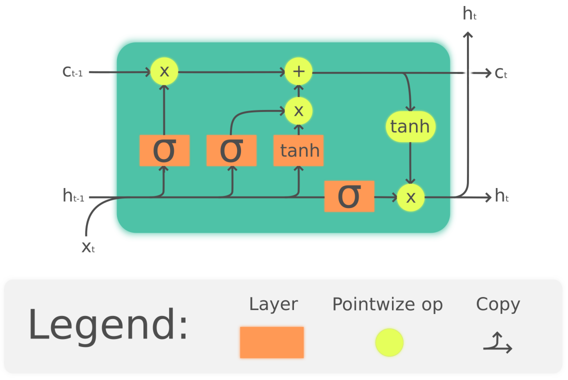


Figure 1. Computation within a single LSTM cell.

3. Approach

In this section, we first give a short summary of the dataset used. This is followed by breaking down the video labelling process into the frame-wise feature extraction step, and the feature sequence labelling step; and explaining each of them.

3.1. THETIS Dataset

THETIS stands for THree dimEnsional TennIs Shots dataset[3]. It is a human action dataset comprising of videos where subjects perform 12 different tennis strokes/actions. Each video has one of the subjects performing one of the 12 actions in front of a camera *without the ball*. Each video is 70-150 frames in length (2-3 seconds). There are a total of 55 subjects (31 amateurs and 24 professionals), performing each of the 12 actions thrice, for a total of 1980 videos. The 12 actions included in the dataset are listed below;

- Backhand with two hands
- Backhand
- Backhand slice
- Forehand slice
- Forehand volley
- Service flat
- Backhand volley
- Forehand flat
- Forehand open stands
- Service kick
- Service slice
- Smash

Figure 2. Different subjects performing different shots in the THETIS dataset.

The original dataset provides the videos in 5 different synchronized forms (RGB, silhouettes, depth, 2D skeleton and 3D skeleton)[3]. For the purpose of this project, we will stick to using just the monocular RGB videos. The motivation behind this is that such a model can easily be tested

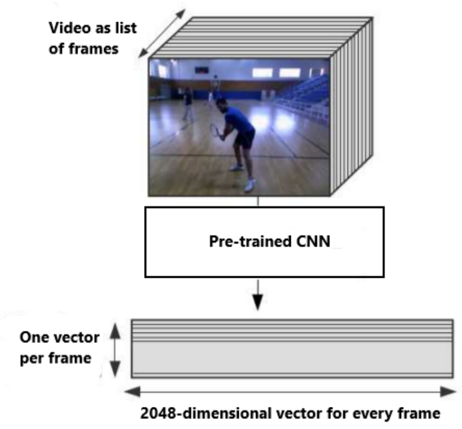


Figure 3. Using a pre-trained CNN to convert video to a list of feature vectors.

upon abundantly available professional game videos with only a little more pre-processing work on those videos (localizing the players in every frame using a pre-built framework). This dataset is particularly challenging for the task of action recognition, because the difference between the individual actions is very nuanced. This problem is further augmented by the dataset having a mix of professional and amateur players, and the absence of the ball from all the videos.

3.2. Converting frames to features

The first step of the video processing is done at the frame level. The main goal of this step is to extract the salient features from every frame of every video. To do this, we first resize every video to a size compatible with the input size of the CNN. We then pass every frame of the video through a pre-trained CNN, and use the activations from the final convolution layer as the features of that specific frame. This is motivated in section 2.1. We perform spatial averaging over this feature in order to get a 1-dimensional vector for every frame, for every video. The dimensionality of this feature depends on the CNN architecture used. In our case, the features are 2048-dimensional. We now have a list of feature vectors for every video, with the length of list being equal to the number of frames in the video. Now to be able to further process these feature vectors with a recurrent network, we need to clip (or pad) every video until each of them has the same number of features. We make every video feature list 100-long by either, clipping frames from the end of the video, or adding blank frames.

3.3. Labelling sequence of features

After completing the above step, we now have 100 feature vectors corresponding to every video in the dataset.

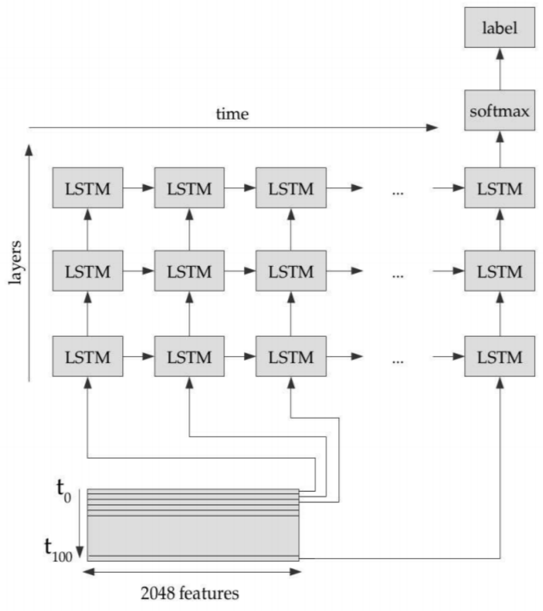


Figure 4. LSTM network for action recognition.

The ordering of the vectors corresponds to the ordering of the frames in the video. Now, this sequence of features can be further processed using an LSTM architecture, designed to take a sequence of inputs and produce a single label for the sequence as output. In our case this label is nothing but the action performed in the video, which is converted to a one-hot vector corresponding to the action label.

The figure 4 shows the LSTM that was used by Mora et al. [8]. Note that this is not the best performing architecture we found. The details of the various methods tested, and their comparison is in the experiments section below.

3.4. Implementation details

This section outlines the details of the 'baseline' implementation, which was the architecture suggested by Mora et al.[8]. We have also tried slight variations of this architecture, and other approaches which are discussed in the experiments section.

- **Pre-processing steps:**

  - Convert video to list of frames. If number of frames is less than 100 do zero-padding, else clip the frames to length 100.

  - Apply a *feature extractor* (Pre-Trained Inception or ResNetV2(50 layered)) on image frames to extract a 2048-dimensional feature vector per frame.

  - Stack features of frames from a video to create a 100×2048 feature block for every video.

- Create a list of all such feature blocks with each block corresponding to a video and create a corresponding list of the labels (the labels here are in form of one-hot vector corresponding to their action label)

- **Train-Validation-Test Split:**

  - The list containing the feature blocks consists of data from 55 people doing 12 actions with each action repeated in three videos.

  - Train split consists of data due to first 40 people.(120 videos per action)

  -Validation split consists of data from people numbered 41 to 50 (30 videos per action)

  - Test split consists of data from people numbered 51 to 55 (15 videos per action)

- **LSTM architecture:**

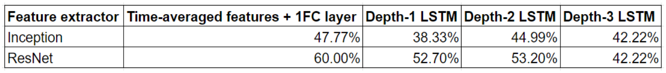
  - The LSTM architectures we tried were of depths 1,2 and 3. The hidden cell size of LSTM was fixed to 90 for all the layers. Every LSTM layer weights were regularized with L2 regulariser with $\lambda = 0.003$.

  - The learning rate used for optimisation was initialised at 0.01 and was decayed with a factor of 0.95 after every 50 epochs.

4. Experiments & Results

4.1. Test Accuracies

The table below shows the test accuracies averaged across all the 12 stroke classes, for the different model types and feature extractor choices.



Feature extractor	Time-averaged features + 1FC layer	Depth-1 LSTM	Depth-2 LSTM	Depth-3 LSTM
Inception	47.77%	38.33%	44.99%	42.22%
ResNet	60.00%	52.70%	53.20%	42.22%

Figure 5. Feature extractor vs model.

From the table above, we can say the following;

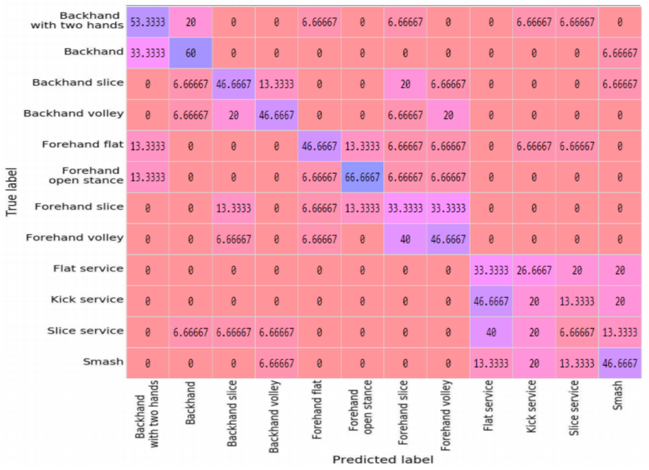
- ResNet is a better feature extractor for this task, as compared to the Inception network, independent of the choice of model architecture.

- Amongst the LSTMs of different depths, depth-2 LSTMs performed the best, irrespective of the feature extractor.

- Time-averaged feature with the ResNet feature extractor gave the best average accuracy amongst all models tried.

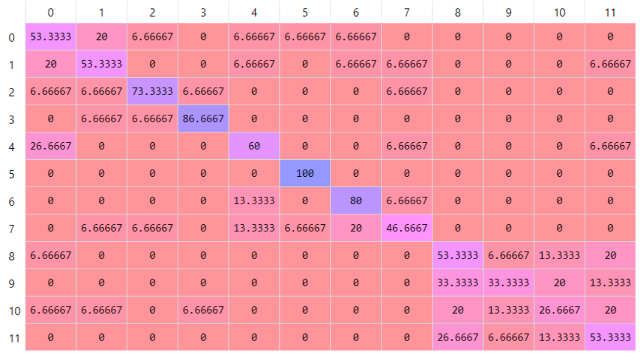
4.2. Confusion Matrix

A good way to visualize where exactly the model is making the most mistakes, is by plotting a confusion matrix across all classes. This will show us as to which of the classes, the model is having the most difficulty classifiying correctly. Below, we show the confusion matricies for 2 of the 8 models we experimented with. The labelling of the rows and columns is consistent across all the confusion matricies, as shown in figure 6. Note that, according to this labelling, the shots/strokes adjacent to each other are similar.



True label \ Predicted label	Backhand with two hands	Backhand	Backhand slice	Backhand volley	Forehand flat	Forehand open stance	Forehand slice	Forehand volley	Flat service	Kick service	Slice service	Smash
Backhand with two hands	53.3333	20	0	0	6.66667	0	6.66667	0	0	6.66667	6.66667	0
Backhand	33.3333	60	0	0	0	0	0	0	0	0	0	6.66667
Backhand slice	0	6.66667	46.6667	13.3333	0	0	20	6.66667	0	0	0	6.66667
Backhand volley	0	6.66667	20	46.6667	0	0	6.66667	20	0	0	0	0
Forehand flat	13.3333	0	0	0	46.6667	13.3333	6.66667	6.66667	0	6.66667	6.66667	0
Forehand open stance	13.3333	0	0	0	6.66667	66.6667	6.66667	6.66667	0	0	0	0
Forehand slice	0	0	13.3333	0	6.66667	13.3333	33.3333	33.3333	0	0	0	0
Forehand volley	0	0	6.66667	0	6.66667	0	40	46.6667	0	0	0	0
Flat service	0	0	0	0	0	0	0	0	33.3333	26.6667	20	20
Kick service	0	0	0	0	0	0	0	0	46.6667	20	13.3333	20
Slice service	0	6.66667	6.66667	6.66667	0	0	0	0	40	20	6.66667	13.3333
Smash	0	0	0	6.66667	0	0	0	0	13.3333	20	13.3333	46.6667

Figure 6. Confusion matrix for our implementation of the model suggested by Mora et al. [8]. This model uses the Inception feature extractor with 3-layer deep LSTM model. Its average accuracy was 42.22%.



	0	1	2	3	4	5	6	7	8	9	10	11
0	53.3333	20	6.66667	0	6.66667	6.66667	6.66667	0	0	0	0	0
1	20	53.3333	0	0	6.66667	0	6.66667	6.66667	0	0	0	6.66667
2	6.66667	6.66667	73.3333	6.66667	0	0	0	6.66667	0	0	0	0
3	0	6.66667	6.66667	86.6667	0	0	0	0	0	0	0	0
4	26.6667	0	0	0	60	0	0	6.66667	0	0	0	6.66667
5	0	0	0	0	0	100	0	0	0	0	0	0
6	0	0	0	0	13.3333	0	80	6.66667	0	0	0	0
7	0	6.66667	6.66667	0	13.3333	6.66667	20	46.6667	0	0	0	0
8	6.66667	0	0	0	0	0	0	0	53.3333	6.66667	13.3333	20
9	0	0	0	0	0	0	0	0	33.3333	33.3333	20	13.3333
10	6.66667	6.66667	0	6.66667	0	0	0	0	20	13.3333	26.6667	20
11	0	0	0	0	0	0	0	0	26.6667	6.66667	13.3333	53.3333

Figure 7. Confusion matrix for the model that performed best during experimentation. This model uses the ResNetV2 feature extractor, and applies a single fully connected layer over the time-averaged outputs of ResNet. Its average accuracy was 60%.

From the above confusion matricies, we can observe the following:

- In both cases, a block of high, non-zero entries occurs in the bottom right corner of the matrix. This shows that both models are facing a difficulty differentiating between the various service types (and smash). This is to be expected because the racket movement is almost exactly same in all those strokes.

- Mis-classification amongst similar strokes is lesser in the second confusion matrix. This can be seen as the entries near the diagonal (but not on the diagonal) are higher in the first confusion matrix, as compared to the second one.

4.3. Accuracy vs LSTM hidden layer size

The table below shows the variation in average validation accuracy w.r.t. the number of hidden layer units in every layer. This is using the Inception feature extractor, with the 3-layered LSTM architecture, run for 200 epochs each.

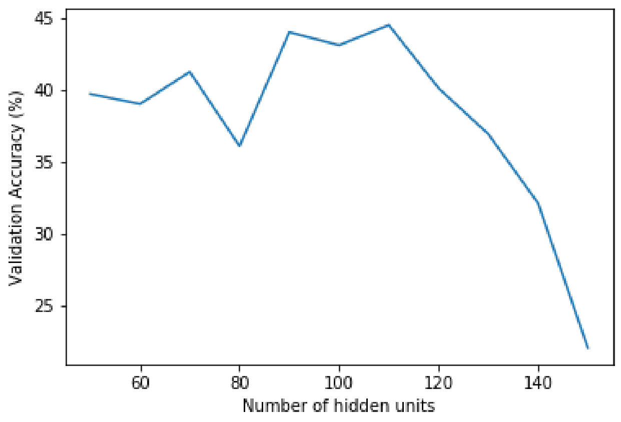


Figure 8. Validation accuracy vs number of hidden units.

From the plot above, it can be seen that using 90 hidden units per layer is the best choice.

5. Conclusions

Open source implementation.
In this project, we have implementated the architecture, and replicated the results from the paper by Mora et al. [8]. Our implementation, along with the code of all the experiments on the action recognition task in tennis, is made available on https://github.com/harshmalara/TennisVideoActionClassification.git for further experimentation. The code is written in python-3, using the tensorflow and opencv libraries. The precise instructions for running the code are available on the link itself.

Improved choice of feature extractor.
In our experiments, we used both, InceptionNet[9] and ResNet[4] trained on the ImageNet dataset, as black-box feature extractors. We observed that ResNetV2 (the 50-layered architecture) was outperforming InceptionNet (used in the original paper [8]), irrespective of the depth of LSTM, and also in the time-averaged case.

Time-averaged features vs LSTM.
This was the most surprising result of this project. The original paper doesn't compare the LSTM's performance with any time-independent baseline. The baseline established by us was using the time-averaged features for a video, being classified using a single Fully Connected layer, followed by a softmax non-linearity. We found that the test accuracy using these time averaged features was better than that of any LSTM architecture. This result was especially surprising as we believed there was temporal information that needed to be learnt before classification could be done and therefore, the time-averaged features wouldn't be able to capture it. We speculate that, either the dataset does not have significant temporal information needed to be learnt for action classification; or average features for different actions are significantly different that they can be easily classified by just a single fully connected layer.

Fine vs coarse grained stroke classification.
In all the confusion matricies formed during the experiments, it was observed that similar stroke types are confused most often. This means that, given the limitations of the THETIS dataset (absence of tennis ball), the model is able to learn the general stroke classes (backhand, forehand and serve) much better than the more nuanced strokes as originally intended in the dataset.

6. Future Work

Player expertise detection.
The dataset contains videos of both, amateur players and professional players. This means that after training the network, the feature vectors at the output of the network for a professional and an amateur player can be compared in order to give an estimate of how close the stroke of the amateur player is as compared to the professional. This can finally be used to give feedback to a user, as to how good or bad his shot is.

Application to game footage.
There is a huge abundance of professional tennis game videos online, in high resolution. However, they are different from the THETIS dataset mainly due to the fact that the player is continuously moving around in a real game, and the camera is not continuously centred on a single player. In order to make this data work with the model above, we need to localize the player across frames and then crop those set of 100 frames in which the player is performing the shot. Once this pre-processing is done, the data on which the above method can be tested will vastly

increase.

Fine-tuning CNNs
In this result, we have only focused on using the CNNs as pre-trained black boxes to extract our features from. One can also study the effects of fine-tuning the CNN together with the new LSTM part, specifically for this task. Also, we were unable to use ResNet152 due to a RAM bottleneck on the colab servers. This is an architecture which we believe can give slightly better performance.

Comparison on other datasets.
In this project we compared the naive, time-averaged CNN features with those learnt by the LSTM and observed that the time-averaged ResNet features were performing slightly better than LSTMs of all 3 depths. We also discussed that this maybe due to the fact that temporal data (ordering of frames) may not be as useful in the case of this particular dataset. Hence, to confirm this, the above LSTM architecture and CNN should be compared on another action recognition dataset, such as the HMDB (Human Motion Database) [6] dataset.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[3] S. Gourgari, G. Goudelis, K. Karpouzis, and S. Kollias. Thetis: Three dimensional tennis shots a human action dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 676–681, 2013.

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

[5] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[6] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: a large video database for human motion recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011.

[7] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[8] S. V. Mora and W. J. Knottenbelt. Deep learning for domain-specific action recognition in tennis. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 170–178. IEEE, 2017.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.