CNN for task classification using computer screenshots for integration into dynamic calendar/task management systems

Anand Sampat
Avery Haskell
Stanford University
{asampat,ahaskell}@stanford.edu

Abstract

We explore classification of computer tasks based screenshot images obtained from various sources with sufficient diversity using deep CNN architectures. Our best CNN can achieve 64% accuracy using a convolutional net with 3 convolutional layers, two of which are pre-trained by the hybridCNN database and the last of which is retrined with 10800 images into 14 classes. On a smaller dataset with only 5 classes and 1900 training images, CNNs prove not as effective. Methods for classifying streaming data with parallel CNNs is shown to yield good long-term classification. Window bounding requires more adavanced algorithms to prove effective for this task.

1. Introduction

Classifying high level tasks (e.g. checking emails, photo editing, social media) a user does on his or her computer with a high accuracy is a simple task for most human beings. However, for computers, this is still an unsolved problem. Deep CNN networks are well known for learning features specific to each class in order to better identify new images. Accurate classification can determine when users switch tasks, allowing computers to track time and ensure optimal productivity of the user. No prior work on the subject of screenshot classification exist, however transfer learning on CaffeNet[1] for objects and Places205CNN and HybridCNN [2] for scene classification, provides weights that significantly reduce training time by placing the solver in a state closer to the optimal state. We explore two datsets one small (1900 training examples), and one larger (10,800 training images).

2. Prior Work

Much of the prior work for convolutional neural networks use large nets and large datasets to learn features to recognize objects and scenes, both of which have many different labels and have complex heirarchal structure [1] [2]. No prior work has been completed on screenshots for task classification, however the task is similar to that of scene classification. Here the scene is the context of the task a user may be doing. Thus the net will look for large-scale global features (instead of local ones). However, as mentioned in [2], a hybrid approach is best. For our dataset, screenshots both consist of different windows which could be different objects as well as global full screen features that identify this as a particular computing environment. Further work on segmented CNNs with contextual cues can is shown to obtain better results as well [1], however our data is simply labelled with only 1 or two descriptors thus we decide to forego this technique.

2.1. Data

The data consists of screentshot images curated from two sources: 1) automated image search from search engine scraping Google, Yahoo, Bing, Baidu and 2) screenshots taken at regular intervals from my own computer. I have obtained two relatively clean datasets - small and large. The goal was to obtain 500-1000 images per category since previous literature shows LeNet can get up to 93% accuracy with handwriting MNIST data with 10000 per class and on purely black and white data. Our data however includes all color data (RGB) and thus has more variation. Images are preprocessed to conform to caffe standards prior to processing (i.e. resized to 227x227x3 and channels transposed). Details for each dataset are given below.

Small Dataset

- Training: 1900; 50 batch
- Validation: 485; 5 batch
- 5 classes - (0-4 in chart)

Large Dataset

- Training: 10,800; 200 batch

Figure 1. First row is example of email images, second row is example of calendar images, and final row is example of word processing images.

- Test: 2720; 20 batch
- 14 classes - (0-13 in chart)

index	name
0	calendar
1	email
2	programming
3	social_media
4	word_processing
5	web_shopping
6	finance
7	news
8	music_editing
9	picture_editing
10	video_editing
11	watching_video
12	web_browsing
13	web_forum

2.1.1 Data Challenges

Three challenges to overcome for the data were data sparsity, a lack of data diversity, and data cleanliness.

1. Data Sparsity: Since the data was mined and curated manually, manual data augmentation by moving around windows in addition to standard default augmentation in Caffe was implemented.

2. Data Diversity: Data from my laptop was too homogenous. For more heterogenereity (e.g. different OS, color scheme, etc.) we scrapped the web and hand curated the data for quality.

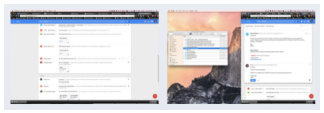

Figure 2. Increased number of images by varying positions of window - manual data augmentation

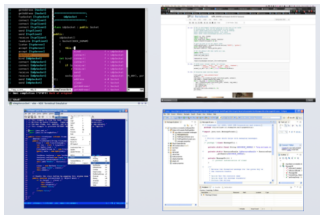

Figure 3. Obtained images from various sources to introduce variation within each class - (e.g. programming as above)

3. Data Cleanup: In order to clean up the data we employed the following steps (in addition to hand curation)

   - Remove duplicates with a d-hashing algorithm (i.e. remove images with same hash)
   - Remove data $<$ 1 kB
   - Remove data $<$ 200 px in H or W
   - Remove data with mobile ratio (i.e. incorrect for desktop) - H $>$ W
   - Remove all images that cannot be processed with PIL
   - Rename all files with class and number (e.g. email_512.JPEG)

2.2. Models

First we baselined our code against simple linear models, i.e. SVM and softmax classifiers. Due to sparsity of the small dataset and higher random chance of guessing, we only baselined the large dataset on these models.

Due to relatively miniscule datasets, our approach was to utilize weights from CNNs already trained on larger datasets. We first tried CaffeNet, which is trained on ImageNet classes and retrained the final softmax layer for 4500 iterations. This model was also used by the Places205-CNN, however additional images and different classes were used for scene classification. Retraining the softmax for this model yielded results worse than random, thus the Hybrid-CNN from [1] was used and a variation with fewer conv layers to reduce overfitting.

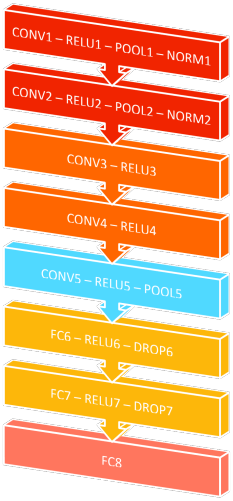

Figure 4. Full hybridCNN model

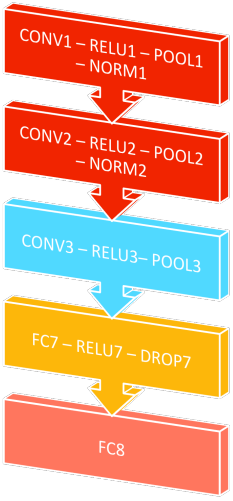

Figure 5. Mini hybridCNN model to prevent overfitting of our dataset.

2.3. Results

2.3.1 Training

All full model and mini model specifications are trained over 10000 iterations. While this is not comparable to the number of iterations in [1] or [2], we empirically notice little change in validation error past this threshold (not to mention training takes at least a few hours).

As seen above small dataset has large overfitting problem - the training accuracy is close to 1 and the gap between training and validation is large. Due to small batch size (5 images) the stochastic variation is high.

The large dataset has less overfitting when the conv3 layer is retrained since retraining just fc8 and fc7 was ineffective.

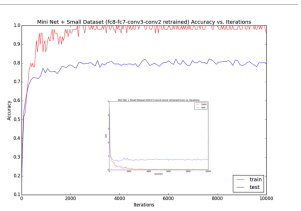

Figure 6. Mini model training on small dataset.

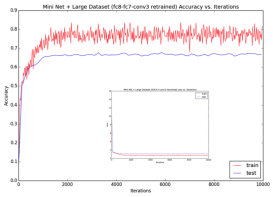

Figure 7. validation error vs. epochs - note this is a simple network and has little data, thus it converges to a poor value quickly.

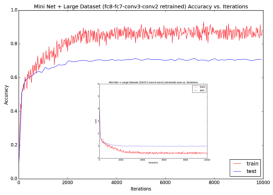

Figure 8. validation error vs. epochs - note this is a simple network and has little data, thus it converges to a poor value quickly.

To improve, we also tried to retrain conv2, however within 10000 iterations the training error increases as does the gap between training and validation. The decreased performance can be seen in section “Testing”.

2.3.2 Testing

Below is a table of results from various models. Note that the we cannot guarantee our linear classifiers and CNNs necessarily have optimal hyperparameters, however reported values are best results from hyperparameter exploration. All test results for CNNs are taken from a sample of 1000 random images within the test set and averaged over a few iterations.

The best models are highlighted in green above. From the table above, we can see the linear classifiers are much better than CNN classifiers for the small dataset. This is expected since we have a small dataset and only 5 classes - however that trend does not continue on to the larger dataset. Thus the result for the full model where only the fully connected layers are retrained, the high testing accu-

Model	Dataset	Layers Retrained	Test Accuracy
SVM	Small	-	0.917
Softmax	Small	-	0.611
Full	Small	fc8,fc7,fc6	0.79
Mini	Small	fc8,fc7,conv3	0.738
Mini	Small	fc8,fc7,conv3,conv2	0.641
SVM	Large	-	0.472
Softmax	Large	-	0.576
Full	Large	fc8,fc7,fc6	0.33
Mini	Large	fc8,fc7,conv3	0.624
Mini	Large	fc8,fc7,conv3,conv2	0.457

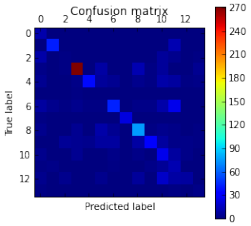

Figure 9. Mini model on large dataset with fc8,fc7,conv3 retrained

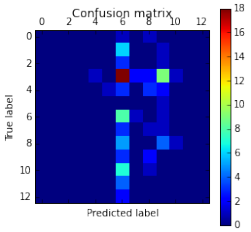

Figure 10. Mini model on large dataset with fc8,fc7,conv3,conv2 retrained

racy follows since simple classifiers are known to do well on this data. For both the small and large datasets, we noticed that retraining two CONV layers reduced performance. We explore the differences between the convolutional layer outputs in "Net Visualizations". The full net does not perform well on the data at least not within a reasonable number of iterations. Note, we didnt retrain any convolutional layers but it would take 10000s iterations to see if it actually works, at which point overfitting is more likely.

Below is the confusion matrix from the two best models on the large dataset. As we will see below, the conv2 retraining changes how the convolutional net sees various structures. The highly regular structure of things like email, calendar, and programming have lower confidences with this model and since the sample size for the large test set tends to have highly irregular classes such as web_forum, music_editing which already have low confidence with the first algorithm, these are consistently misclassified as the class finance likely due to the variation in finance-related

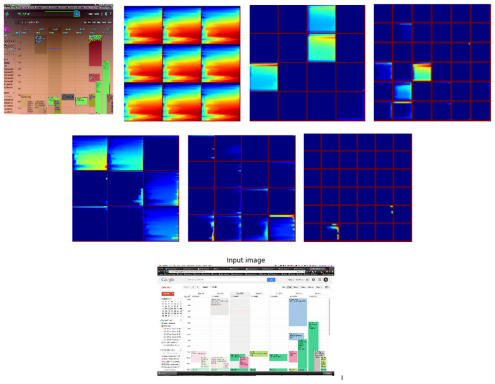

Figure 11. Maximum magnitude filters for given layer. The bottom net has a much larger confidence. From left to right, data layer, conv1, conv2, conv3. Top: Retrained conv3 and conv2. Bottom: Retrained conv3 only.

tasks (e.g. bar charts, pie charts, line graphs, excel spreadsheets, etc).

3. Net Visualizations

We look at the two top models (highlighted in green in the table) and their outputs for each convolutional layer. Note that we were not able to use the deconvolutional approach used in [6], however the outputs of each layer can still show how each layer incrementally improves on the classification. Below we highlight some of the convolutional layer outputs and briefly explain them. The activation of all convolutional layers is extremely varied for the top net. The bottom net is more conservative and highlights the top of the image (similar to email) while downplaying the rest of the image. The bottom net also tends to highlight relevant parts of the image such as the rightmost calendar event which is indicative of a calendar vs. email for example. This shows that without even training on screenshots, already the hybridCNN weights can better identify regions of the image, whereas the top image has very uniform filter outputs. In both cases we see the filter picks out more specific spots. This is more obvious in the bottom net where the whole image, then the top, and then the events themselves are identified. We might expect this to be better as we train the net for more iterations. For email as calendars we see when we retrain more convolutional layers, the layers tend to activate more based on the images it is given. Here, this translates to looking at the full image which lowers the confidence of email based images that have different sizes for example. In particular, the filter below does a better job of picking certain "hotspots" such as the top of the image, which is more image agnostic and better at generalizing. Curiously, this follows similarly to the calendar which results in the filters choosing features fuch as the header and

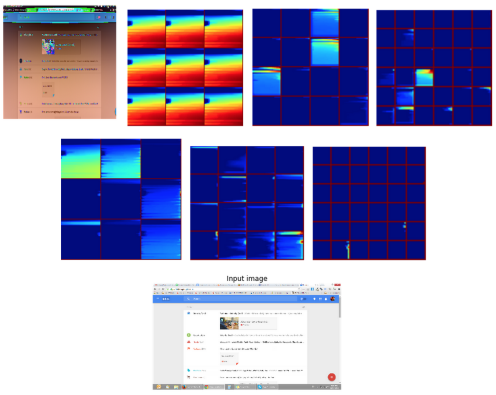

Figure 12. Maximum magnitude filters. The bottom net has a much larger confidence. From left to right, data layer, conv1, conv2, conv3. Top: Retrained conv3 and conv2. Bottom: Retrained conv3 only.

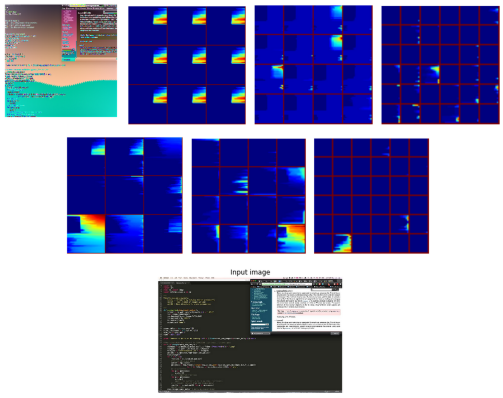

Figure 13. Maximum magnitude filters. The bottom net has a much larger confidence. From left to right, data layer, conv1, conv2, conv3. Top: Retrained conv3 and conv2. Bottom: Retrained conv3 only.

sidebar. Note how by conv3 the bottom has already identified specific areas while the top is still unsure of where to look. Again the bottom layer is more selective. Programming images tend to be dark (usually a terminal window), which is shown by the majority unactivated filters, however, while this is a feature, the top net emphasizes the presence of a box in the top right which is rare and not indicative of programming. Conversely the bottom net has filters that look all around the box without emphasizing the top right resulting in a higher confidence. Once again by conv3 the bottom has discovered that the top right is for the most part useless, however the top still is significantly biased by it.

3.1. Window Detection - Data Preprocessing

Users tend to multitask when using computers. In order to better assess this we try a method to automatically detect

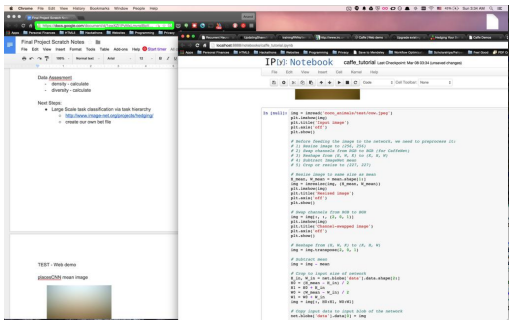

Figure 14. Bounding box captures an area too small (no other rectangular contours were detected).

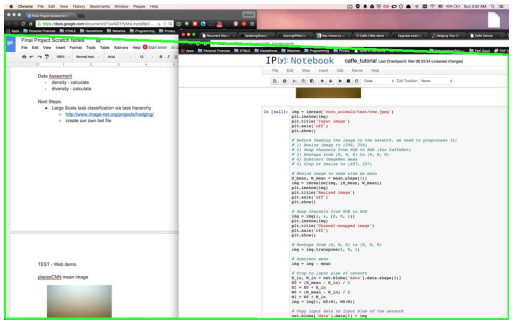

Figure 15. Bounding box captures an area too large and while quadrilateral, it doesn't capture the rectangle we expect.

the most important window in the image when the window is not already full screen. Object detection here would be overkill due to the regular nature of windows (i.e. rectangular), therefore, instead of using the CNN, we use a canny edge detector with OpenCV [5] to find 4 edges that form the largest rectangle in the image. In addition an R-CNN approach may detect objects within the windows themselves confounding the analysis.

The detector first applies a Gaussian filter to smooth noise caused by gradients from color changes etc. Then the intensity gradients are calculated using the formula:

$$\mathbb{G} = \sqrt{\mathbb{G}_x^2 + \mathbb{G}_y^2}$$
$$\Theta = \arctan(\mathbb{G}_y, \mathbb{G}_x)$$

The results are not ideal. The majority of boxes formed do not capture the windows themselves regardless of the edge detector used (Canny edge detection, differential edge detection, and phase congruency edge detection).

3.2. Streaming Data - Different input layers

Two methods of "data streaming" were explored. The first uses window of images and averages their result before inputting the image into the CNN. The next inputs all frames into a CNN independently and then averages output

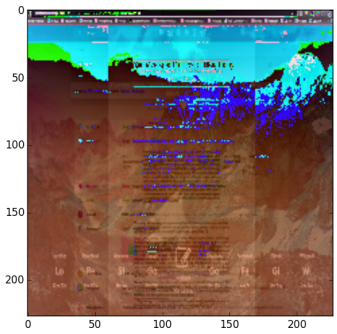

Figure 16. Example of averaged images over the window. Classifier get's confused - artifacts cause mislabelling

probabilities to obtain the final classification. The dataset used is realtime images taken at 10 second intervals and the window size is 4. Here we aim to capture the idea that temporal context is important to classification. That is, a user who switches many times between tasks within a given window likely is not switching tasks. The dataset is 20 images that span two classes - email and web browsing. The method of averaging images fared much worse and misclassified 95% of images while the average probabilities only misclassifies 20%. We conclude that apart from computational overhead, parallel CNN's and probability averaging is a better strategy.

3.3. Conclusion

On a large dataset our best model performed well with 64% accuracy using a CNN with weights from hybridCNN training and retraining the conv3, fc7, and fc8 layers on 10,800 images . In order to better integrate this however, data streaming is very important. We have proven streaming is effective when using parallel CNNs, however future work is needed to assess feasibility of running multiple CNNs in parallel for more frequent sampling (e.g. every second). Previous work on video classification proves this high sampling can be tractable, however, bundling of frames and classification into a large corpus labelled data (a la the Sports-1M dataset) is required [6]. With such a training set, we can host the network on EC2 and allow the network to run in test mode on anyone's data. Furthermore, obtaining contextual cues via segmentation of the image with bounding boxes can be used in conjuction with a contextually constrained deep network to improve task labelling via the method in [1].

4. References

1. T. Kekec, R. Emonet,E. Fromont,A. Tremeau,and C. Wolf "Contextually Constrained Deep Networks for Scene Labeling". [http://www.bmva.org/bmvc/2014/files/paper032.pdf]

2. B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. "Learning Deep Features for Scene Recognition using Places Database." Advances in Neural Information Processing Systems 27 (NIPS), 2014 [http://places.csail.mit.edu/]

3. A. Krizhevsky,I. Sutskevar, G. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems 25 (NIPS) 2012 [http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolut pdf]

4. J. Long,E. Shelhamer, and T. Darell. Fully Convolutional Networks for Semantic Segmentation.[http://arxiv.org/pdf/1411.4038v1.pdf]

5. http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html

6. M. D. Zeiler and R. Fergus. "Visualizing and Understanding Convolutional Networks. [http://arxiv.org/pdf/1311.2901v3.pdf]