Using Siamese CNNs for Removing Duplicate Entries From Real-Estate Listing Databases

Sergey Ermolin
Stanford University, Stanford, CA
sermolin@stanford.edu

Olga Ermolin
MLS Listings, Inc, Sunnyvale, CA
oermolin@mlslistings.com

Olivier Moindrot, Stanford University , CA **

Rishi Bedi, Stanford University , CA **
Boya (Emma) Peng, Stanford University , CA **

Abstract

Aggregation of geo-specific real-estate databases results in duplicate entries for properties located near geographical boundaries. The lack of nation-wide listing identifiers makes it hard to weed out the duplicates. This paper presents an approach of identifying duplicate entries via the analysis of images that accompany real-estate listing leveraging transfer learning Siamese architecture based on VGG-16 CNN topology

1. Introduction

1.1. Real estate databases are geo-specific (eg. East Bay, North Bay, South Bay, etc). If a house to be put up for sale is located close to the geo boundary, a real estate listing agent will often list it in both databases. For example, a house located in Milpitas would often be listed in both East Bay and South Bay databases. The content of both database entries could be different to appeal to different demographics of each area. Real estate brokerage firms do enter in cross-area sharing agreements and there are efforts underway to create a nation-wide sharing framework as well. Herein lies the problem: when data feeds from EastBay and SouthBay databases are aggregated, this results in two duplicate listings. The purpose of the project is to provide means to identify and flag these duplicates for future removal.

1.2. Contrary to what one might think, property's street address by itself is not enough to identify the duplicate entries as it is often misspelled or even misrepresented to make the house appear to belong to a more desirable city (eg: Almaden vs San Jose or Antelope vs Sacramento). Some of the most reliable indicators of duplicate listings are jpeg images uploaded by agents, but these can't always be simply binary-compared since they may contain broker watermarks, be cropped, flipped or have post-processing visual effects such as color modifications.

** CS231N TA

2. **Previous work**.

Most of the inspiration for this approach was drawn from Chopra et.al [1] where the Siamese architecture involving CNNs was first described. The structure of CNNs, introduction of the last fully-connected layer and loss function definitiontion were leveraged from subsequent works [2] and [3]. We also considered Triplet architecture described in [4], but decided against using it because it did not seem to offer higher accuracy for the application at hand.

3. Technical approach

3.1. We are re-using the Siamese architecture described in [1] and [3]

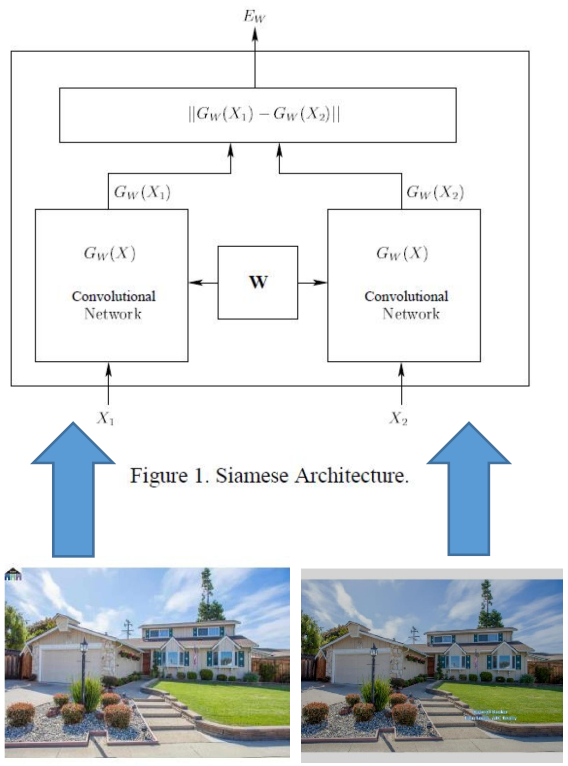


Figure 1. Siamese Architecture.

In the original paper [1], the authors described the outputs Gw(X) as "two points in a low-dimentional space". For our implementation, we chose to use vgg-16 convolutional network, but modified its topology following the suggestion in [3] by removing the last layer and softmax loss and instead adding an extra fully-connected "FC_extra" layer. The conceptual implementation is shown on Figure 2.

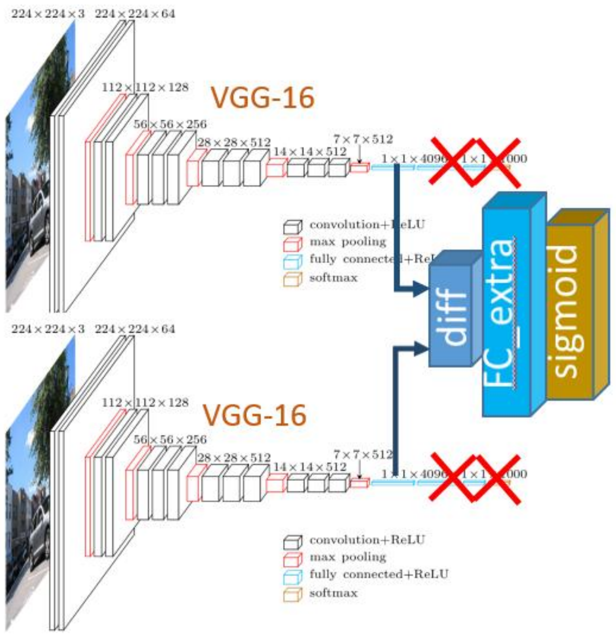


Figure 2. Conceptual implementation of VGG-16-based Siamese architecture.

The additional "FC_extra" layer combines 4096-dim difference between feature outputs of two branches of Siamese networks into a single-value output which is then classified using a logistics regression.

3.2. **Convolutional Network Topology**. While initially we were planning on using Inception_v3 topology, it proved to be difficult to overcome Tensorflow syntax when importing it. Fortunately, Olivier Moindrot provided an excellent example of doing a transfer learning with vgg-16 network [8]. Based on advisor's feedback that for the problem at hand both vgg-16 and Inception should provide similar accuracy, we switched to vgg-16 topology. The last fully-connected layer of vgg-16 network [vgg-16/fc8] was reducing 4096-wide feature vector to a scalar output. Since we needed to preserve a wide feature vector, we removed [vgg-16/fc8] layer and fed outputs of two [vgg-16/fc7] layers to "difference" function implemented as a fully-connected layer. This last layer essentially implements the formula below as suggested in [3] (shown here for L1 (absolute value) difference norm).

$$\left(\sum_j \alpha_j \left|\mathbf{h}_{1,L-1}^{(j)} - \mathbf{h}_{2,L-1}^{(j)}\right|\right)$$

3.3. **TensorFlow model implementation.** On advice of Olivier Moindrot, we simplified the code and computational complexity by implementing just one VGG-16 network instead of two while feeding it twice as many images. The actual implementation is shown on Figure 3

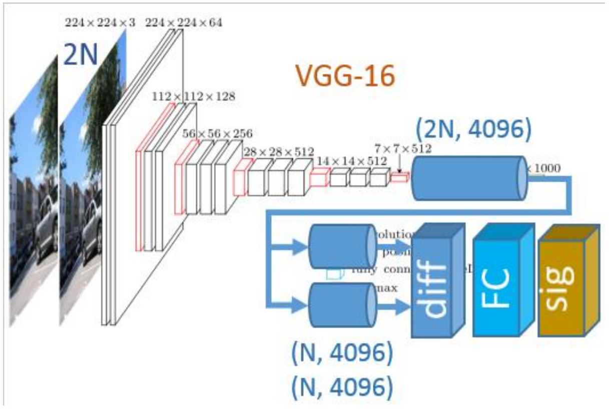


Figure 3. Actual implementation of VGG-16-based Siamese architecture in TensorFlow rev1.2rc0

3.4. **Loss function and "difference" norm**. Chopra et. al strongly argued in [1] against using a square norm and in favor of using an L1-norm (abs[h1-h2]). We tried both L1 and a square norm (L2) and found L2 to result in better accuracy and precision. Thus, in the end, we decided to use L2 even though [3] used an L1 norm (absolute value of the difference). The accuracy and precision difference are summarized in Table

Difference norm used	Accuracy	Precision
L1 (abs)	0.62	0.59
L2 (square)	0.75	0.98

Table 1. Accuracy and Precsition for L1 and L2 feature difference norms.

3.5. **Activation function**. After experimenting with softmax cross-entropy function which seemed to be an over-kill for this 2-class logistics regression problem, we finally settled on using a sigmoid function.

3.6. **Optimization**. The original skeleton code provided by Olivier Moindrot enabled us to first train just newly-added layers and only then train the entire network starting with pre-trained VGG-16 checkpoint weights. We reused and extended this approach by adding precision/recall measurements as well as test_data scoring run. We started with a plain SGD and were able to achieve accuracy on the order of 70%. We have also tried Adam optimization which took a while to get working in tensorflow because it required initializing local variables, but found that it did not perform significantly better than traditional SGD, so we revered back to SGD and were able to achieve good results by lowering the learning rate and adjusting dropout which was used for the published results. As expected, the majority of the time for this project was spent debugging TensorFlow code, preparing dataset, but mostly optimizing hyperparameters. In particular, since we needed to deal with two different learning rates (one for training the newly-added layers and one for overall model training), we found ourselves spending three times as much effort on this task as we expected to. Other challenging optimization task was experimenting with different image sizes, random flipping and cropping of training and validation images which was also affecting training accuracy. Perhaps somewhat surprisingly, weight decay and dropout rate did not seem to significantly affect loss and accuracy values. Perhaps if we spent more time adjusting the learning rates, we would have reached a point where the effects of other hyperparameters became more pronounced.

4. **Dataset**.

4.1. We obtained a curated image dataset provided by MLS Listings which contains entries (sets of jpeg images) that are a-priory known to belong to duplicate real estate listings as well as those which were distinct. We started debugging on a micro-dataset (~100 images), then proceeded to a larger one (~500 images) and did the final evaluation on a dataset containing a total of ~3500 images, both identical and duplicates. Below are some of the examples of images from the dataset that would be considered “Duplicates”

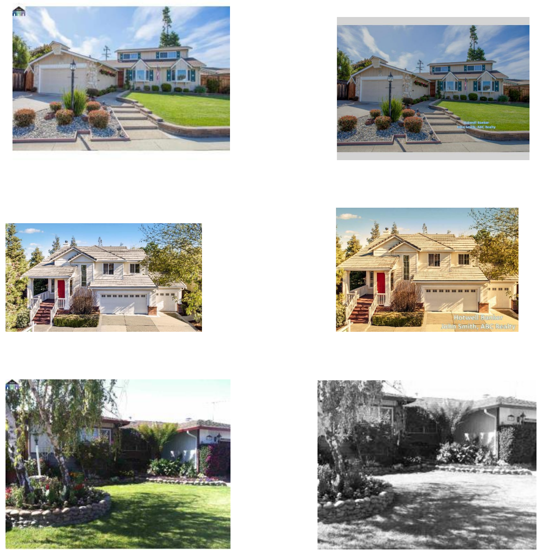


Figure 3. Sample images from the dataset provided by MLS Listings.

4.2. For duplicate pairs, we used both identical images as well as those that real estate agents altered by changing size, resolution, color correction or watermarking. We did exclude from the dataset the images of the same object (eg. front of the house), but photographed from different angles. An example of the excluded pair of duplicate images is shown below in Fig. 4

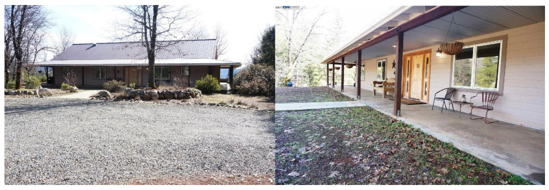


Figure 4. Example of two images of the same house feature which were excluded from the dataset.

While some broker watermarking was large and prominent (see Fig 3, upper right image), other watermarking was tucked in a corner of the image and barely noticeable. See Fig 5 below.

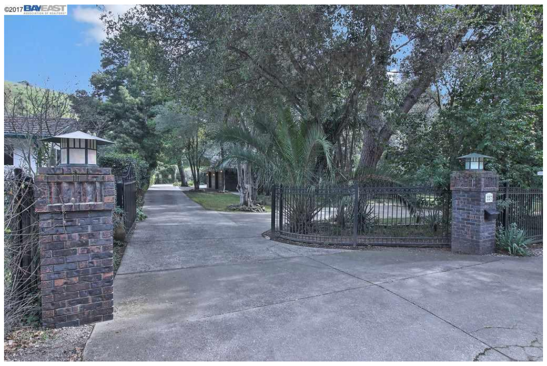


Figure 5. Image with minimal watermarking added in the upper left corner.

All images were of various resolution. As part of image preparation stage of the pipeline, we wrote a simple python PIL script that resized all images to 244x244.

4.3. **Dataset Structure.** Along with a directory containing images, MLS Listings provided us two types of datasets:"Duplicates" and "NoDuplicates". Each dataset contained a directory of images and a CSV file of the following format:

'image_filename_1, image_filename_2, label'

"Duplicates.csv" file would contain only labels = 1, indicating duplicate files, and have N lines, eg:

```
15192704-1.jpeg,   81600251-1.jpeg,  1
15193792-1.jpeg,   81595903-1.jpeg,  1
……………………………………………
```

"NoDuplicates.csv" file would contain both '0' and '1' labels, with '1' appearing only for the identical filenames. The file would have C(N, 2) lines, eg:

```
15419889-1.jpeg,   81639585-1.jpeg,  0
15508107-1.jpeg,   15508107-1.jpeg,  1
```

To construct the training dataset, we would merge the 'NoDuplicates' and 'Duplicates' directories together and concatenate their respective .csv files. We would then randomly shuffle lines in .csv files and sequentially draw the desired number of filename pairs and corresponding for training set starting from the top of the .csv file, while drawing validation set starting from the end of the file. We would make validation set size = 20% of training set size.

4.4. To avoid having a large dis-balance of similar and dissimilar items in the dataset, we would also randomly remove entries with label=0 until the ratio of 0/1 labels would be somewhere between 30% and 70%.

4.5. We also constructed a small (50 images) holdout test dataset that was not used during model training. After every model run, it was used for the final scoring.

5. Experiments/Results/Discussion

5.1. As a primary success metric we used Accuracy which is defined as (TP+TN)/Total, where TP – "true positive" and TN – "true negative". In addition, for the test dataset, we also calculated Precision = TP/(TP + FP) and Recall = TP/(TP + FN), where FP – "false positive" and FN – "false negative". We would also monitor the loss function during training.

Table 2: Metrics definition

Category	Prediction	Label
TP	1	1
TN	0	0
FP	1	0
FN	0	1

Table 3: Metrics definition

Metric	Formula
Accuracy	(TP + TN)/Total
Precision	TP/(TP + FP)
Recall	TP/(TP + FN)

5.2. **Baseline.** Before embarking on building a convolutional neural network, we attempted a brute-force approach using a 1-Nearest Neighbor algorithm borrowed from CS231N assignment-1. We had to slightly modify our definition of the "labels" in order to make it work in the following way. Each image in our dataset of K images was assigned a label corresponding to its "similar" twin. Those without a twin, we assigned label 'K+1'. For example, the array of labels for 7 images would look like this:

[1,0,3,2,8,8,8]
which would mean that in the array of images:
- Images 0 and 1 are twins
- Images 2 and 3 are twins
- Images 4, 5, 6 are unique.

For a test subset of 33 images, the distance matrix would look like Fig , where the darker shade indicates similarity. Not surprisingly, diagonal squares are black indicating distance=0.

When training, we would first run our network with all VGG-16 weights "frozen" except for FC7 and the added FC_distance layer for several epoch (~5 epoch turned out to be sufficient as the loss function would stop improving afterwards). Then we would continue the run allowing all layers to be trained for 5-20 epochs, depending on the size of the training dataset.

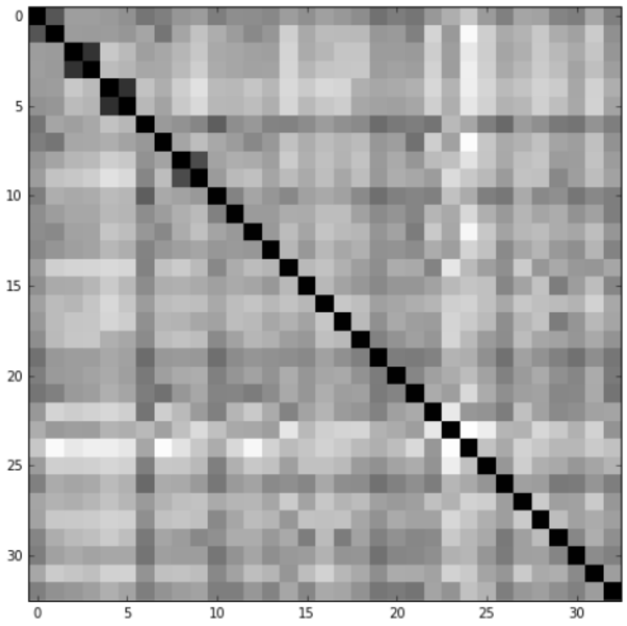


Figure 6. Distance matrix representation for 1-Nearest Neighbor implementation.

By eye-balling the matrix, one can see that images 2/3 and 4/5 are very close together, while images 6/7 and 10/11 are far apart. The actual image pairs are shown on Fig 7 and Fig 8.

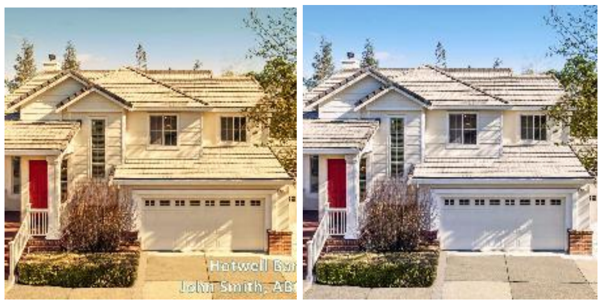


Fig 7. Images 2 and Image 3

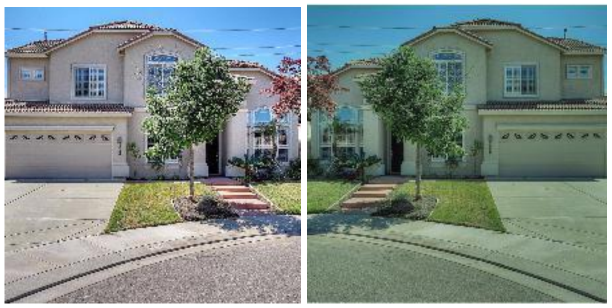


Fig 8. Images 10 and Image 11

One can see (perhaps not surprisingly), that while nearest neighbor approach could cope to some extent with watermarking and color editing, it completely failed when images were flipped or significantly cropped.
On a larger dataset, overall precision of Nearest Neighbor approach was found to be 24.4% which is in line with previous work on KNN algorithms.

5.3. **VGG-16 CNNs.** As a sanity-check, we periodically run our model on a very small dataset (80/20 train/val) and making sure that it would overfit the data. Typical results of the sanity-check run would look like this (table 4)

Epoch	Tr_acc	Val_acc	
1	0.2	0.2	Training the last two FC layers only
5	0.5	0.4	Full training
10	0.85	0.4	Full training
20	0.85	0.4	Full training

Table 4. Overfitting a small training set.

For the full dataset, both training and validation accuracy would increase with the number of epochs. Here are our final results in table 5

Epoch	Tr_acc	Val_acc	Test
1	0.2	0.2	
5	0.66	0.63	
10	0.75	0.69	

20	0.76	0.69	Accuracy = .65
			Precision = 0.92
			Recall = 0.66

Table 5. Final training and testing results.

5.4. After 10 days of parameter optimization and trying two algorithm, this is a typical shape of our loss function (fig 9)

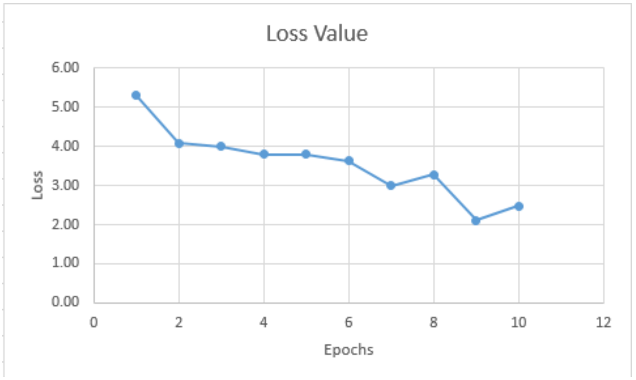


Fig 9. Typical loss function

5.5. We found the fact that the precision between different runs was 85-95%, indicating that there were very few false positives, i.e. the model would rarely mark two different test images as being the same (FP). It would, however, more often make a mistake of classifying two identical images as being different (FN), which is probably due to the fact that the images were cropped to 224x224 prior to being presented to the network.

5.6. **Validation approach.** Since we enjoyed having access to a large dataset (far larger than we could process in a reasonable time on a single GPU available to us), we chose not to do K-fold validation, but instead drew data randomly from available dataset and for both training and validation.

5.7. **Test data approach.** For testing holdout dataset, we selected a 50 images which we carefully reviewed to make sure they represented a reasonable approximation to real estate photo listings found in practice.

5.8. **Test data analysis.** We carefully analyzed True Positives and False Positives responses of our network. We found that the network not only correctly recognized images on Fig 7 and 8 above (just like a Nearest-Neighbor network), but could also identify flipped image as being identical – something that the Nearest-Neighbor approach failed to do. Somewhat surprisingly, Siamese network consistently identified the two test images shown on Figure 10 as different, while Nearest-Neighbor network did not have such difficulties. Trying to figure out the reason for it could be a topic for future work.

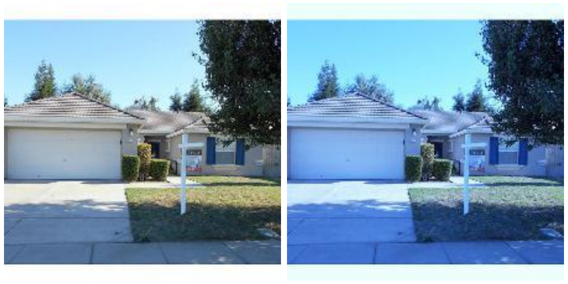


Figure 10. Image_5 and Image_6.

6. Conclusion/Future work

6.1. Our implementation of Siamese network was able to achieve a respectable validation accuracy of 65% with precision of close to 100%. This was about 40% higher than the brute-force Nearest Neighbor approach and allowed to correctly classify heavily cropped or flipped images. In [3], the authors quoted accuracy of about 90% with training sets of 30k – 150k. Our training set was under 5k, so a lower accuracy was not all that surprising.

6.2. As far as future work is concerned, we would like to spend more time optimizing the network a bit further, as well as rewriting the code to map it to a large Apache Spark cluster in order to speed up the computation. This would allow us to do performance benchmarking of single-node GPU vs a cluster of CPUs.

Careful analysis of prior work also revealed that some authors often use RoC (True positive rate plotted vs True netagive rate) as a quality metric for such work. Given enough time, we would like to add such a metric to our analysis

References

[1] Chopra, S., Hadsell, R., LeCun, Y.: Learning a similarity metric discriminatively, with application to face verification. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 1, pp. 539–546 (2005). ISBN 0769523722

[2] S. Bell, K. Bala: Learning visual similarity for product design with convolutional neural networks. ACM Trans. Graph. (TOG), 34 (4) (2015), p. 98

[3] Koch, G., Zemel, R. S., & Salakhutdinov, R. (2015). Siamese neural networks for one-shot image recognition. In ICML Deep Learning Workshop.

[4] E. Hoffer and N. Ailon. Deep metric learning using triplet network. CoRR, /abs/1412.6622, 2015

[5] Ji Wan, Dayong Wang, Steven Chu Hong Hoi, Pengcheng Wu, Jianke Zhu, Yongdong Zhang, and Jintao Li, "Deep learning for content-based image retrieval: A comprehensive study," in ACM Multimedia, 2014, pp. 157–166.

[6] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In Proc. CVPR, 2015.

[7] Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In Proc. Computer Vision and Pattern Recognition Conference (CVPR'06). IEEE Press.

[8] tensorflow_finetune.py by Olivier Mondroid. https://gist.github.com/omoindrot/dedc857cdc0e680dfb1be99762990c9c

[9] KNN algorithm implementation of CS231N HW assignment-1. http://cs231n.github.io/assignments2017/assignment1/ , http://cs231n.stanford.edu/assignments/2017/spring1617_assignment1.zip

[10] Fei-Fei, Li, Fergus, Robert, and Perona, Pietro. One-shot learning of object categories. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 28(4):594– 611, 2006.

[11] Taigman, Yaniv, Yang, Ming, Ranzato, Marc'Aurelio, and Wolf, Lior. Deepface: Closing the gap to human-level performance in face verification. In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, pp. 1701–1708. IEEE, 2014.

[12] Kumar, B., Carneiro, G., Reid, I.: Learning local image descriptors with deep siamese and triplet convolutional networks by minimising global loss functions. arXiv preprint arXiv:1512.09272 (2015)

[13] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. (2012) 1097–1105

[14] E. Hoffer and N. Ailon. Deep metric learning using triplet network. arXiv preprint arXiv:1412.6622, 2014.

[15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015, pages 448–456, 2015

[16] J. Masci, D. Migliore, M. M. Bronstein, and J. Schmidhuber. Descriptor learning for omnidirectional image matching. In Registration and Recognition in Images and Videos, pages 49–62. Springer, 2014

[17] VGG-16 Tensorflow model. http://download.tensorflow.org/models/vgg_16_2016_08_28.tar.gz

[18] Tensorflow v1.2rc0. 1.2.0rc0 here: https://www.tensorflow.org/versions/r1.2/install/

https://www.cs.toronto.edu/~frossard/post/vgg16/vgg16.png

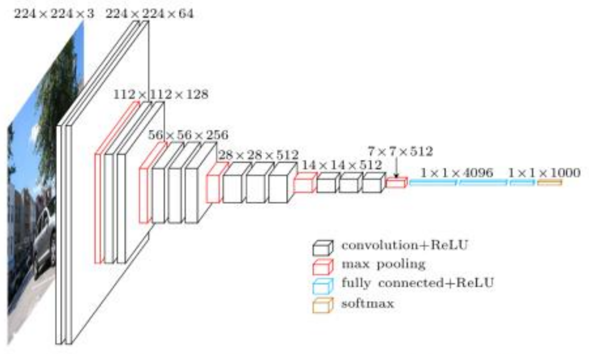
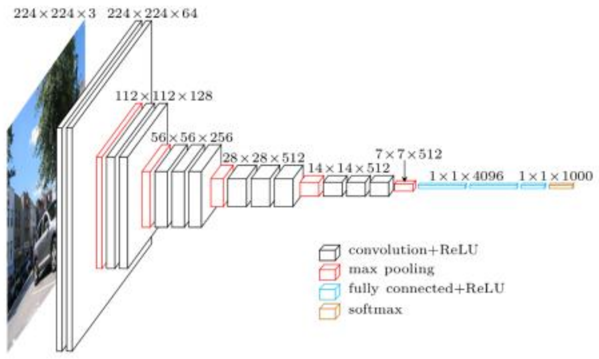


FIG. 2 - MACROARCHITECTURE OF VGG16[5]



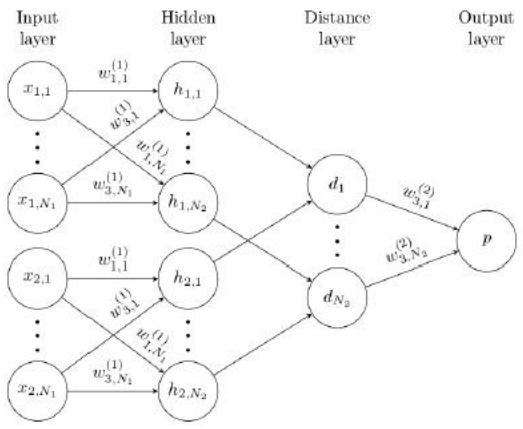


Figure 3. A simple 2 hidden layer siamese network for binary classification with logistic prediction p. The structure of the network is replicated across the top and bottom sections to form twin networks, with shared weight matrices at each layer.