New lightweight invariants for image recognition

Abstract.

This is the first of a three-part investigation into fast-to-calculate functions from the image pixel position and intensity to an array of numbers where the function is invariant or close to invariant for the image or image segment even after similarity or perspective transformations. The goal for this first phase is to investigate using these invariants to improve a CNN's accuracy to recognize images; improve transfer learning by relying on fewer images; and, tune parameters for the lightweight invariant.

Fast calculation is one essential advantage not provided by alternative invariants under similarity transformations such as Zernike polynomials or Hu image moments as well as others. This part of the investigation suggests a hierarchy of invariants and a near invariant for perspective transformations which is not supported by the above alternatives.

Overview.

An essential characteristic of image recognition of an object in a picture is that the object is recognizable as the same object regardless of where on the picture that object is located. It must also be recognizable if it is rotated or dilated or contracted as well as any translation. In technical terms, the object is still recognized after any similarity transformation. Finally, we would like to recognize it after a perspective transformation that has the viewer at a different point in space.

Recognition of an object can be facilitated by any value that is constant i.e. invariant after any similarity transformation of the object in the picture. Even better is if it is invariant after a perspective transformation. The constant value can be a single real number or an array of real numbers that is calculated from the pixels in the picture and do not change after any similarity transformation.

The book "Deep Learning with Python" (2018) by Francois Chollet states that one of the important strengths of Convolutional Neural Networks (CNNs) is (page 123):

> *The patterns they learn are translation invariant*. After learning a certain pattern in the lower-right corner of a picture, a convnet can recognize it anywhere for example, in the upper-left corner. A densely connected network would have to learn the pattern anew if it appeared at a new location. This make convnets data efficient when processing images (because *the visual world is fundamentally translation invariant*): they need fewer training samples to learn representations that have generalization power.

Even though a CNN takes advantage of translational invariance of an image, a CNN needs data augmentation by rotating and dilating/contracting the object to recognize other similarity transformations. A CNN also needs augmentation to recognize perspective transformations. The need for this extra load of training samples might be reduced with the use of similarity invariants such as what is described next.

Ideas: lightweight invariants and their hierarchy.

The primary focus of the investigation is on the invariants constructed from the number of shaded in pixels between concentric circular shells, aka donuts, around the centroid of the object. The construction of the invariant assumes we pick shells whose radius is a fixed percentage of the distance to a fixed point on the image (such as the point that is furthest from the centroid). With that assumption, define the similarity invariant to be the number of shaded in pixels divided by the total number of shaded in pixels in the image. This is invariant to similarity transformations.

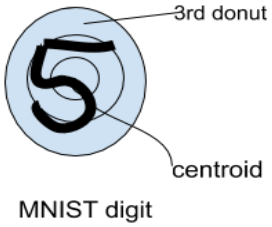

Figure 1. Shells about the centroid of an image.

For example, consider four concentric circles whose radii are, respectively, ¼, ½, ¾ and 1 multiplied by the distance between the centroid and the shaded in pixel that is farthest from the centroid. Then the invariant under similarity transformations is four numbers: the number of shaded in pixels in each of the four donuts divided by the total number of shaded in pixels in the image.

A hierarchy of Invariants can be built on top of this invariant. For example, the centroids of the image inside each donut form a set of points. Call points A and B the centroid of the image piece contained in the first and second donut, respectively. Let N be the number of donuts. Then there are N centroids and (N – 1) lines from A to the other centroids. Also, these (N – 1) lines form (N – 1) angles with the line AB, one of which is 0 because the angle of the line AB with itself is 0. The tan() of these angles are (N – 1) real numbers. Likewise, for the (N – 1) lines from B to the other centroids. I claim (but without proof in this whitepaper) that these 2*(N – 1) real numbers are invariant to similarity transformations. Figure 2 will help show that this is the case.

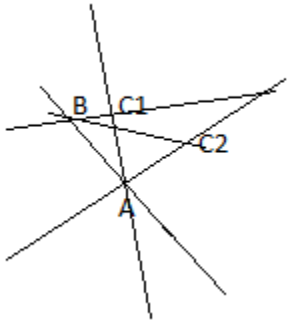

Figure 2. N (=4) centroids and the angles between them.

Note that the angles BAC2, BAC1, BAB (which is 0) and ABA (which is 0), C2BA, C1BA uniquely determines the shape up to a similarity transformation.

If N is 9 then the invariant 16 real numbers summarize a great deal of the structure of the image. This is as opposed to the 784(=28*28) numbers of a 28x28 pixel image. This is fast-to-calculate because tan() of the angles is just the slope which is fast-to-calculate since it is fast to get the coordinates of shaded in pixels.

Other invariants can be constructed in an analogous way by using the information of the relative size of the donuts. For example, the radii that we pick could be to make the donuts have equal area instead of radii that are ¼, ½, ¾, or 1 multiplied by the largest radius.

In the future, we investigate using the average array of values for the same label as an average invariant to see if that improves image recognition accuracy.

The invariants described above can be compared with a hash function of an image of an object. They are analogous in that they are quick to calculate and have a low collision frequency with the invariant/hash of the image of a different object. They are unlike each other in that slightly different objects have invariants that are numerically close to each other whereas hash functions map even slight differences to very different results.

Near-invariants under investigation.

A near-invariant is a function that is numerically close to its value after a similarity transformation. One example of a near-invariant is if we draw spokes from the centroid and consider the number of shaded in pixels in each sector of each of the donuts between concentric circular shells around the centroid of the object. If we have many sectors and the image changes gradually from one donut sector to its neighbors, then a rotation will take a sector, roughly, to another sector.

Therefore, a function such as raising to a power the number of shaded in pixels in each sector of each of the donuts and adding them together will roughly be close after a similarity transformation. Note that if the power the function raises it to is 1 then we have the actual invariant described in the previous section because we are just summing the sectors of the donut which equals that of the entire donut.

A suggestion for a fast-to-calculate near-invariant for perspective changes is based on noting that the circles are mapped to ellipses for a perspective change. Donut sectors of the circle are mapped to donut sectors of the confocal ellipses because each line after a perspective change is mapped to another line, so the boundaries of the sectors are mapped to the boundaries of sectors.

Now, the sectors that contain or neighbor the major axis of the ellipse have the least distortion from the circle sectors that the training data uses. The is especially true if the sectors are thin slivers. Since there are four sectors that neighbor the major axis we thus have the part of the similarity vector that corresponds to those sectors of the test image that should be close to the part of the similarity vector that corresponds to those sectors of the concentric circles of a training image.

Thus, we have the pattern where some pieces of the similarity vector of the test image that should be close to the similarity vector of the train image. This is something we can search for. In fact, we also have the pattern that sectors of the test image will be increasingly distorted from of the those of the train image for an entire quadrant. In the next quadrant, going clockwise, the distortion will be

decreasing with each sector. Then in the third quadrant, the distortion will be increasing with each sector. So, the pattern is increasingly distorted, then decreasingly distorted, then increasingly distorted, then decreasingly distorted.

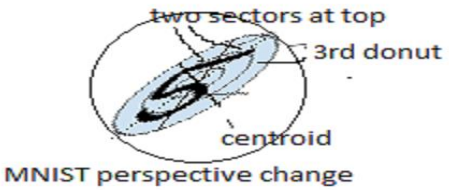

Figure 3. Perspective change and two sectors that are least distorted.

This is something that is reflected in the similarity vectors of the test image and that of a training image that we are comparing against the test image. So, we can search training images or their similarity vectors to see whether this training image fits the two patterns described above.

Phase 1: experiments and results

Phase 1 suggestions in this whitepaper except the hierarchy of invariants and the perspectivity near invariant are implemented. The current version of the code and detailed results of experiments are on the GitHub page's jupyter notebook at https://github.com/scottcohenGA2019?tab=repositories. The lack of the hierarchy of invariants is significant because that is where we would expect the most dramatic improvements.

Phase 2 and phase 3 ideas are described below but are not yet implemented.

Phase 1 investigation of invariants of an image is to append them to the image from the csv file. Phase 1 uses the MNIST dataset, the "hello world" of machine learning algorithms. So instead of a CNN recognizing a 28x28 pixel image, it trains and tests to recognize a 30x28 pixel image where the additional 2 rows are the 32 bytes of the invariant calculated from the image plus 24 bytes of zeros padding to fill out the last row.

There, however, are several reasons why image recognition of MNIST digits would *not* improve with the use of invariants. The digit images are all centered, occupy most of the picture space, and are upright. Therefore, none of the images are translations, rotations, or dilations/contractions of each other. The images are viewed from the top so perspective views from a different point in space do not occur. Therefore, we should expect nothing dramatic for phase 1 results.

Phase 1 compares results to a baseline of a CNN without any use of invariants. The code refers to this as a verbatim test because the implementation is copied verbatim from the book "Deep Learning with Python" (2018) , Francois Chollet page 122.

Another baseline implemented is a CNN recognizing an invariant that is random numbers appended to the image. It would be worthwhile to compare such a baseline to using a dropout layer. Appending

random numbers to the image has a significant improvement of the model accuracy. It might take fewer resources such as CPU than a dropout layer and is therefore worth investigating.

10-fold stratified cross-validation tests are repeated 5 times. There is often a significant stddev() to the results which is why 50 observations are done for each experiment.

Experiments are repeated for a dataset training size of 12, 24, 36, 48 and 60 thousand. This is to see the impact of having more images to train against.

Experiments on model accuracy are done to tune configuration parameters defined in the code: variations on the similarity vector; the number of donuts; the number of sectors for each donut. No experiments are done for CNN or other model hyperparameters.

<results depend on tests that are running now so boxplots of the error reduction rate at different dataset sizes will be given next week>

Future investigations.

Phase 1: Remaining work.

1. Do performance analysis:
    a. speed up get_similarity_vector (avoid python's append() function; use Cpython; vectorize more calculations; use profiler)
    b. tune scikit-Learn hyperparameters and Config. constants
2. Make a module that you can import and pip install.
3. Visualization:
    a. box plots for (1 - accuracy) error rate reduction
    b. AUC-ROC curve for points <tp rate,tn rate> by taking one Config. constant and varying it
    c. are there uses for scatterplots? heatmaps? line graphs?
4. Better documentation:
    a. provide docstrings.
    b. document alternative attempts e.g. USE_AREA_INVAR; the weaker model of kMeans and their centroids

Phase 2 ideas: using invariants with different ML models and datasets.

Different datasets such as COIL from Columbia University or ImageNet will provide images that might benefit much more from invariants than the MNIST dataset because they have images that are translations, rotations, or dilations/contractions of each other. Phase 2 of the investigation will use these datasets in addition to "augmenting" the MNIST dataset with images that we create to shrink the image of the digit and then translate and/or rotate it to a different part of the picture.

An invariant can be used with a Decision Tree based model just as easily as the CNN used it in phase 1. Phase 2 will see how the accuracy when using XGBoost. The explanatory ease of and speed of fitting the data to a Decision Tree based approaches are advantages that makes this investigation worthwhile.

Phase 2 will also try other transformations such as adding smoke or noise to test image. It will also implement and test the invariants for perspectivity transformation.

For completeness, we compare kNN accuracy which is equivalent to comparing the accuracy of just the invariant vector by itself without being appended to either a CNN or XGBoost recognized image.

The method of incorporating the similarity vector into a CNN can be improved. In Phase 1, we simply appended the similarity vector to the image and had a CNN recognize the larger "image". In Phase 2, we investigate incorporating the recognition of the image with its similarity vector into a CNN by following the methods in "INVARIANT BACKPROPAGATION : HOW TO TRAIN A TRANSFORMATION -INVARIANT NEURAL NETWORK" (2016) by Sergey Demyanov, James Bailey, Ramamohanarao Kotagiri, Christopher Leckie.

Phase 3 ideas: building more image recognition on top of invariants.

Again, from the book "Deep Learning with Python" (2018) , Francois Chollet states another important strength of CNNs (page 123):

> *They can learn spatial hierarchies of patterns*. A first convolution layer will learn small local patterns such as edges, a second convolution layer will learn larger patterns made of the features of the first layers, and so on. This allows convnets to efficiently learn increasingly complex and abstract visual concepts (because *the visual world is fundamentally spatially hierarchical*).

Improving recognition of spatial hierarchies of patterns can be achieved as follows: start by looking at each pixel in the image as the centroid of some object. Now, draw concentric circles around a given pixel and look for images whose invariant is close to invariant with each concentric circle that is drawn. This is a massive use of the suggestions in this investigation which is possible only because they are fast-to-calculate.

We can pretrain similarity vectors of many thousands of images or parts of images such as a human smile, a cat smile, etc.; and, put their invariants in a file organized as some tree structure. We could quickly search that tree of vectors for what pretrained image is closest to the target image.

The tree structure would contain the similarity vector as the data as well as keys of all images that contain object such as a human face for the human smile, or a cat face for the cat smile. The human face or the cat face would be the 1st level in the hierarchy of objects that contain leaf level target image.

By repeating the above steps, we branch further and further up the hierarchy of images of the objects that contain the initial object. Eventually we reach as far as we can go and decide it is a cat.

Depending on how close the match is, we might go back down the tree structure to compare other features of the face such as eyes as well as going up the tree structure. The eyes might distinguish the face as that of a cat not a human. If not then the whiskers will.

The above steps are parallelizable and should be amenable to a map-reduce implementation. The tree structure might be done better as a network structure.