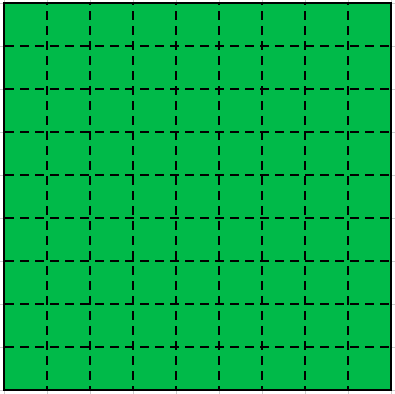
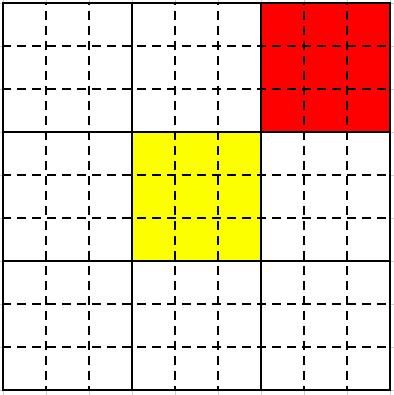
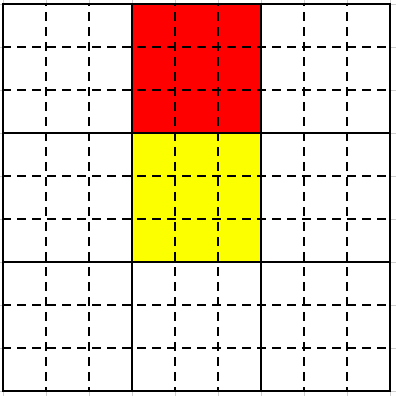
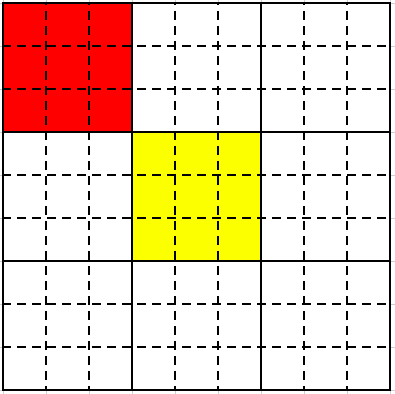
Convolution Weave:

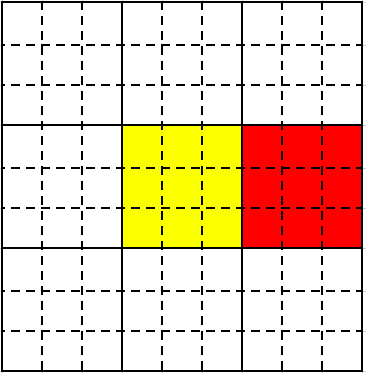
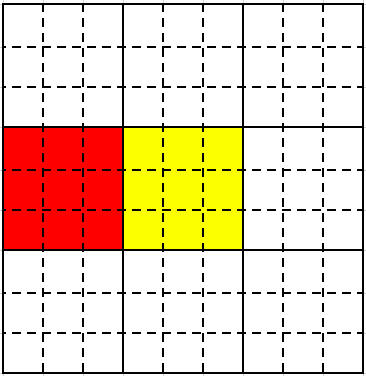
The original purpose of weave convolution was to increase contextual information without increasing the size of the filters. Instead of increasing the size of the filters, the use of two separate passes of filters allows for a large decrease in the number of parameters.

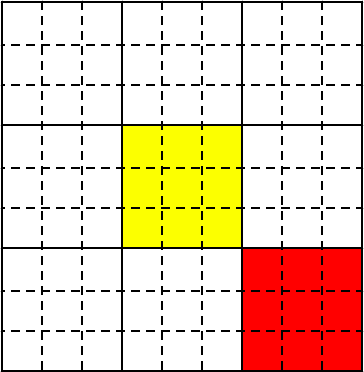
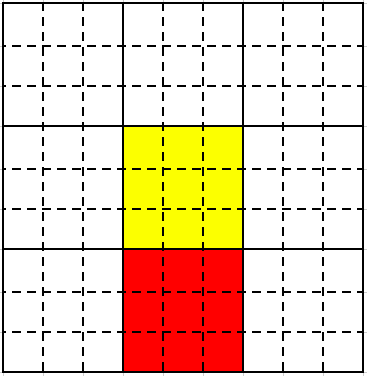
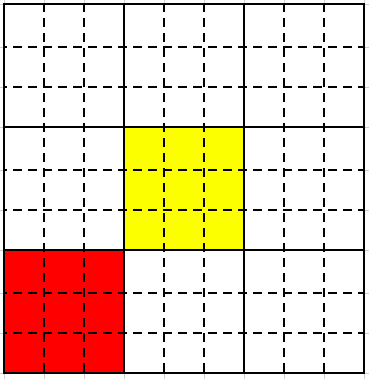
For example, a 9 by 9 filter, with 81 parameters over an image would grant contextual information over that 9 by 9 space.



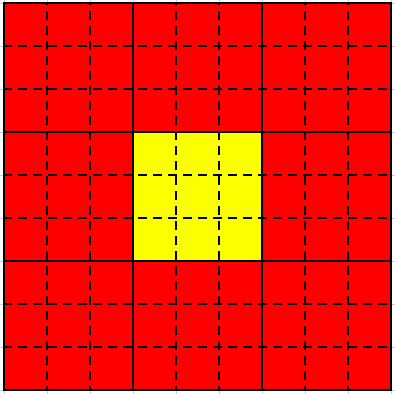
Or we could use 2 sets of 3 by 3 filters, with 18 total parameters over an image in the following fashion all the way around the center pixel. The red representing one filter being applied in 8 locations and the yellow another filter being applied in the center.



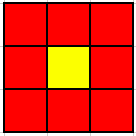




This would look like this all together;



Which results in a 3 by 3 grid which must be convolved on again to result in 1 pixel similar to the 9 by 9.



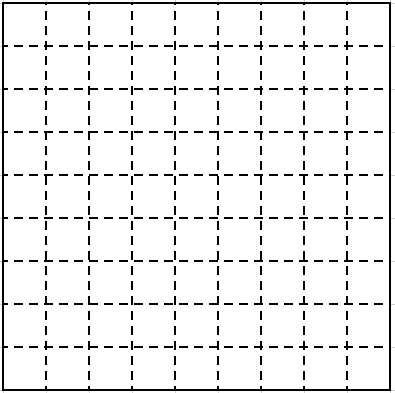
Thus, we can use 27 parameters to gain something similar to the 9 by 9 grid using 81 parameters.

I offer the interpretation that; the first set of filters, the yellow, will be used as local information, and the second, the red filter being repeated 8 times, will be used as peripheral information, or the contextual information.

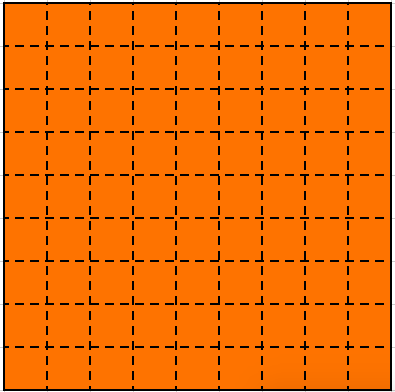
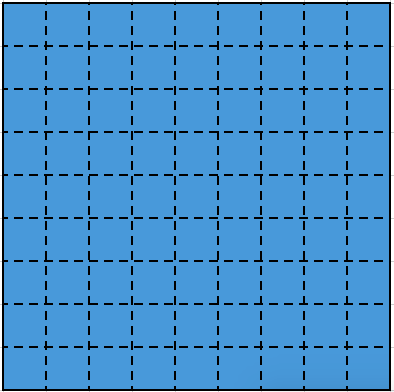
Unfortunately, it is difficult to preform this operation with prebuilt TensorFlow functions so instead we must pass two separate sets of filters which generates two separate arrays. Then the generated arrays will be “woven” together to create a larger image, in a similar fashion as above for each pixel, which will then be convolved on once again.

**An example of this being preformed on a “full image”:**

Let the image below be representing the interior of an image (therefore all filters that extend off the shown section will by definition still in the image to negate the need to show any padding). Below is a 9 by 9 section of an image. (This operation has been detailed with odd size sections to simplify the examples, but is fully defined for even size images.

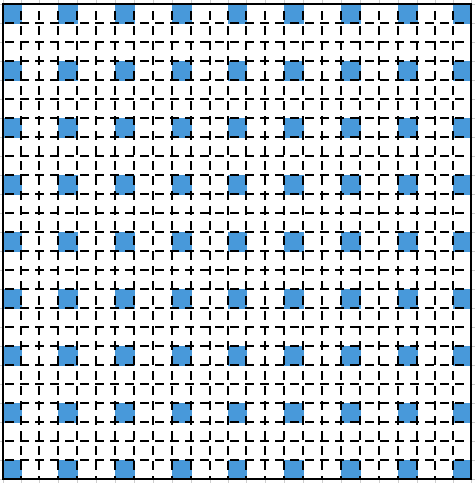


Passing two sets of 3 by 3 convolution filters with stride 1 over the original image section results in two separate arrays of the same size (ignoring the edge of the image which were padded with the appropriate number of zeroes);



Since one of the arrays represents the local filters and the other represents the contextual or peripheral filters the arrays must be operated on differently. For the blue array, representing, the local filters, we must expand this image by 3 times, inserting 2 zero rows between each row and 2 zero columns between each column (due to the filter size of 3 by 3). This operation is detailed in a naïve way (not how it is implemented in TensorFlow) at the end of this document in **ZeroExplode**.

This operation results in an image that looks like this;

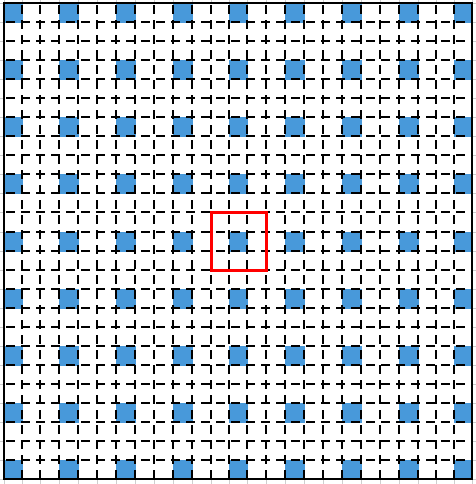


This operation properly spaces each local filter so that there is room for the peripheral filters to be inserted in the proper space.

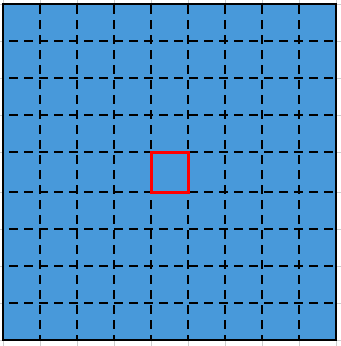
Now for the more complicated step which again is detailed in a naïve way in **ArrayWeave** at the end of this document. We must arrange the following peripheral filters in fashion that surrounds each local filter with the neighboring peripheral filters.

This will be shown for the center pixel, and, in addition, it can be seen that the neighboring peripheral filters are often shared for many local filters.

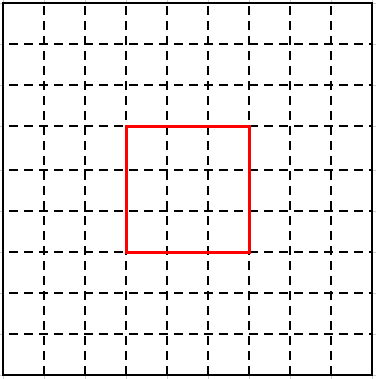
For this center pixel the red square highlights which white pixels must be filled in with the necessary orange filters.



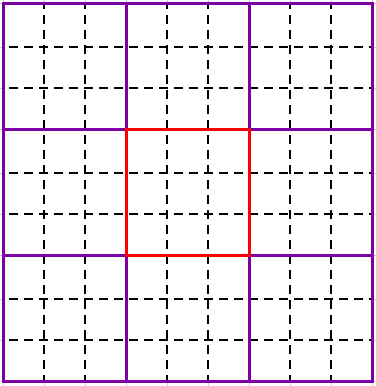
If we look at where this pixel came from we can easily make sense of where the neighboring peripheral filters should come from. The local filter in this red square is shown below.



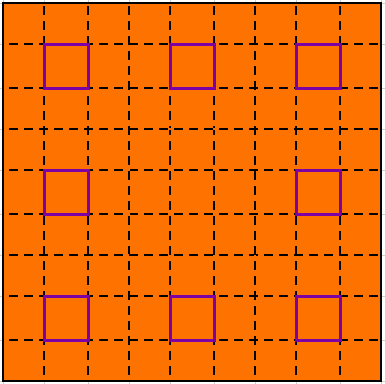
Which comes from the first convolution step:



Now we can see where we want the peripheral layers to come from shown in purple:

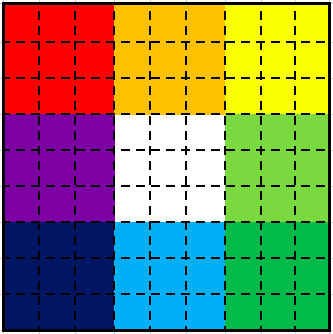


These purple filters correspond to the following spots in the peripheral filter array:

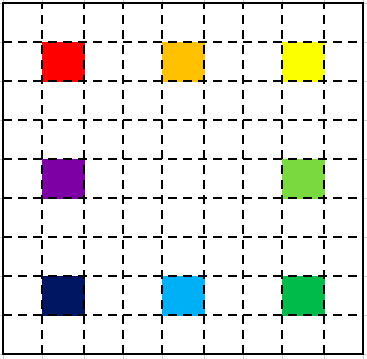


Therefore, it can be seen that we can’t simply add our two arrays together. We must arrange this array in the correct format! Which would eventually result in an array that includes both peripheral filters and local filters woven together. Here we color the peripheral filters from above to help show where they moved and were they are duplicated. Each new color represents a peripheral filter generated from the 3 by 3 filter being moved to the designated locations.

These 8 colored squares represent the regions the original 8 regions that are considered peripheral to the center white square which is a local filter.

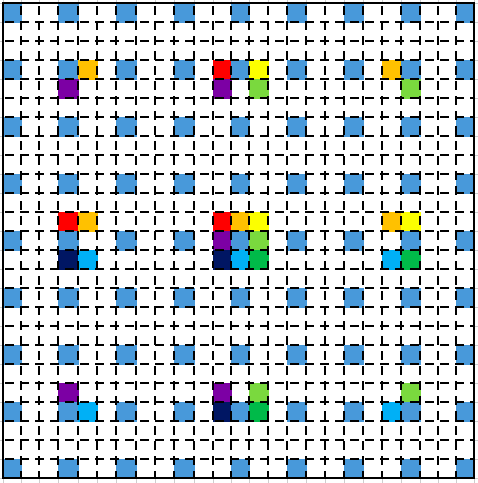


These collapse and are placed into the peripheral array at these positions.

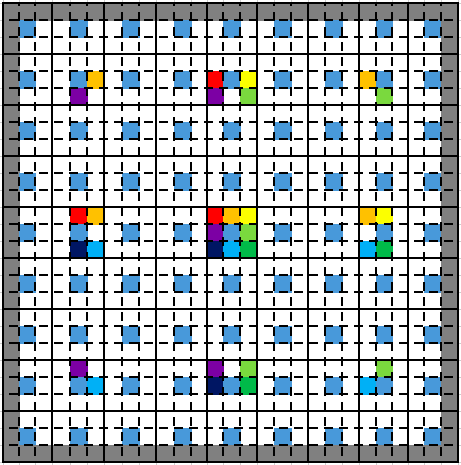


These filters would be placed in the following positons in the local filters. This shows how the operation is being preformed for the center local filter and its 8 neighbors. This operation is preformed for each local filter as if it was the center. Squares of the same color represent the same filter being placed in multiple places.

If a pixel is near the edge of the image and does not have peripheral filters those spots are left blank. Otherwise, every spot is filled with a filter.



This image is then convolved over again with a 3 by 3 filter with stride 3. Here we will show the zero padding to help show where the filters preform their operation. This will result in an array of the same size as the original image.



Below are the algorithms necessary for Weave Convolution. Naïve versions of ZeroExplode and ArrayWeave are presented. Faster versions in TensorFlow exist and in general require creating a dummy array in this fashion and creating a general mapping function that quickly preforms Array and Zero Weave instead of requiring slow loops.

**Weave Convolution:**

**Inputs:**

Image of Shape (*C, N, N*); *image*

Filter Size: *filter\_size* (must be odd)

Number of Filters; *n\_filters*

**Do:**

*pad* 🡨 (*filter\_size – 1) / 2*

*p\_image* 🡨**2DZeroPad(***image,* *pad***)**

*p\_image* of size (*C, N + pad, N + pad)*

*conv\_loc* 🡨 **2DConvolve** *p\_image* with *n\_filters* of size (*filter\_size, filter\_size)* with stride 1

*conv\_loc* of size (*n\_filters, N, N*)

*conv\_per* 🡨 **2DConvolve** *p\_image* with *n\_filters* of size (*filter\_size, filter\_size)* with stride 1

*conv\_loc* of size (*n\_filters, N, N*)

e\_conv\_loc 🡨 **ZeroExplode** *conv\_loc* by 2 \* *pad*

e\_conv\_locof size (*n\_filters, 2\*pad\**(*N-1) –* 2 \* *pad, 2\*pad\**(*N-1) –* 2 \* *pad*)

*w\_conv\_per* 🡨 **ArrayWeave** *conv\_per* by 2 \* *pad*

w\_conv\_perof size (*n\_filters, 2\*pad\**(*N-1) –* 2 \* *pad, 2\*pad\**(*N-1) –* 2 \* *pad*)

*conv\_total* 🡨 e\_conv\_loc + *w\_conv\_per*

*total\_image* 🡨**2DZeroPad(***conv\_total,* *pad***)**

*o\_image* 🡨 **2DConvolve** *total\_image* with *n\_filters* of size (*filter\_size, filter\_size)* with stride

*filter\_size*

*o\_image* of size (*n\_filters,* *N,* *N*)

**return** *o\_image*

**ZeroExplode:**

**Inputs:**

Image of Shape (*C, N, N*); *image*

Distance: *distance*

**Do:**

*e\_image* 🡨 zero array of size (*C, distance\**(*N-1) –* distance, *, distance\**(*N-1) –* distance)

**for** *i\_pos*  in **{**0, 1, …, *N-1*}:

**for** *j\_pos* in **{**0, 1, …, *N-1*}:

*e\_image*[: , (*distance+*1)\**i\_pos*, (*distance+*1)\**j\_pos*] 🡨 *image*[: , *i\_pos, j\_pos*]

**return** *e\_image*

**ArrayWeave:**

**Inputs:**

Image of Shape (*C, N, N*); *image*

Distance: *distance*

**Do:**

*e\_distance* 🡨 2 \* *distance* + 2

*w\_image* 🡨 zero array of size (*C, distance\**(*N-1) –* distance, *, distance\**(*N-1) –* distance)

**for** *i\_pos*  in **{**0, 1, …, *N-1*}:

**for** *j\_pos* in **{**0, 1, …, *N-1*}:

**for** *i\_change* in {-e*\_distance, 0, e\_distance*}:

**for** *j\_change* in {-e*\_distance, 0, e\_distance*}:

*new\_x* 🡨 *i\_pos*\*(1+*distance) + i\_change*

*new\_y* 🡨 *j\_pos*\*(1+*distance) + j\_change*

**if** *w\_image*[*new\_x, new\_y*] exists **and** (*i\_change, j\_change) !=* ***0***:

*w\_image*[: , *new\_x, new\_y*] 🡨 *image*[: , *i\_pos, j\_pos*]

**return** *w\_image*