**Summary on ShuffleNet**

Sources Links: 1. <https://arxiv.org/pdf/1707.01083.pdf>

2. <https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728>

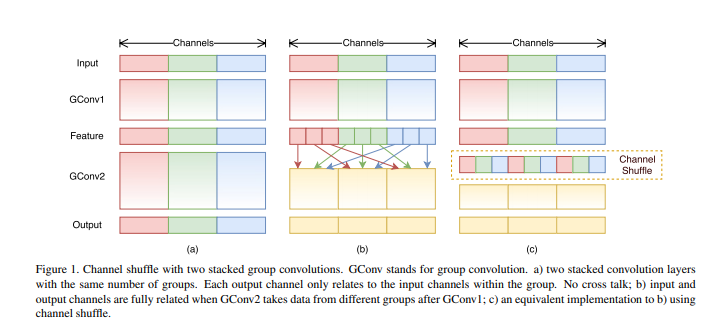
In this paper, the authors developed an extremely efficient CNN which they called ShuffleNet. The network itself was designed to run on devices with little computing power through the inclusion of new operations called pointwise group convolution and channel shuffle. Using the ImageNet classification and MS COCO (common objects in context) datasets, the researchers found that this new architecture performs quite well and is comparable to AlexNet in terms of accuracy while being substantially faster.

Up until this paper in 2017, most research in the field of convolutional neural networks dealt with very large and deep CNNs to solve image recognition tasks. These CNNs would require computation on the scale of billions of FLOPS (floating point operations per second). The creators of ShuffleNet examined the other end of the computing spectrum--looking at a neural network that could operate on the scale of tens or hundreds of MFLOPS that could be used on drones, robots, and smartphones. Before this, pruning, compressing, and low-bit representation were used to reduce the computing power required for CNNs. One of the major hurdles in reducing computation power was finding a way to reduce the complexity or need of 1x1 convolutions found in networks like Xception and ResNeXt. The method proposed to solve this was a channel shuffle operation that would allow for more feature map channels and would enhance performance on very small networks.

The “gold standard” at the time was a network called MobileNet that utilized depth-wise separable convolutions to computationally cheap neural networks. These convolutions consisted of spatial separable convolutions and depthwise separable convolutions. As per source 2, a spatial separable convolution is what the name implies: separating one convolution into two. This method takes one kernel, say 3x3 and separates it into a 3x1 and 1x3. This reduces computation drastically as instead of doing 9 multiplications, now there are only 6. It is important to note that not all kernels can be split into two separate kernels to achieve the same effect as the original 3x3 kernel.

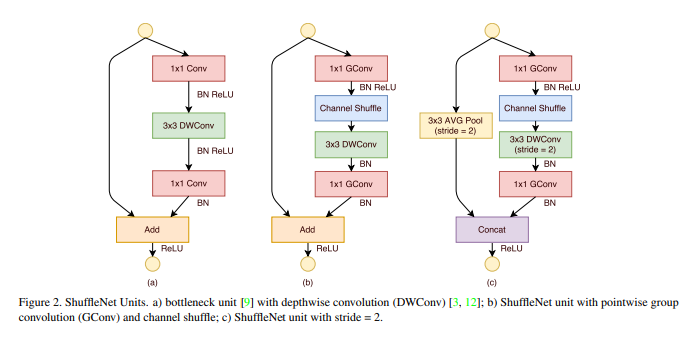
Depthwise separable convolutions, on the other hand, deal with kernels that cannot be split into two smaller kernels. These convolutions split the original convolution into multiple convolutions each dealing with a depth, i.e. channel, and a pointwise convolution. Let’s stick with the 3x3 convolution example. In a depthwise convolution we use 3 different kernels of shape 3x3x1 (rather than a 3x3x3). In other words, a separate convolution for each “depth” (channel) of the network. Pointwise convolution uses a 1x1 kernel to iterate through every point and collapse the depth of the network. In our example this would be a 1x1x3 that would collapse into a shape of 1x1x1. Combining both depthwise convolutions and pointwise convolutions *drastically* reduces the number of multiplications, thus speeding up the network.

Next the paper next examines previous works and methods they will implement in their network architecture. These methods include group convolutions (discussed previously), a **channel shuffle operation** which was rarely used prior to the paper but in the CNN library cuda-convnet, and model acceleration through pruning connects, channel, and redundant connections in a pre-trained model. Below is a figure demonstrating the channel shuffle operation.



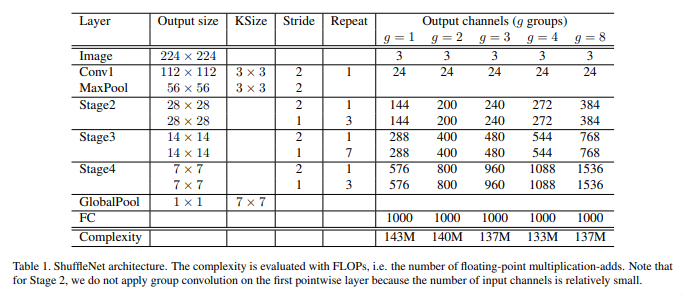
While the researchers examined the architecture of previous state of the art networks such as Xception and ResNeXt for their tradeoff between performance and computational cost, the designs fail to fully account for the cost of the 1x1 convolutions. The researchers rectified this by applying channel sparse connections (i.e. group convolutions also on 1x1 layers) to drastically reduce computational cost. They do note that if these layers are stacked the output is only derived from a small portion of input channels. This can be solved by shuffling the channels, thus allowing for crosstalk between the input and output channels (figure 1c). Figure 1b represents a similar method in which each different input channel is fully related to the output channel.

This is where the paper proposed the ShuffleNet unit based on the channel shuffle operation. The paper proposes that a 1x1 pointwise convolution used in conjunction with a channel shuffle forms a ShuffleNet unit (pictured the figure 2b below). They state that second pointwise group convolution is to restore the channel dimension to match the shortcut path. They tested adding a second pointwise shuffle after this layer, however, the results were largely unchanged. In many architectures I have seen so far, a ReLU typically follows a convolutional layer, but you can see that they do not include an activation after depthwise convolution. To introduce more complexity at the expense of minor computation cost, the model architecture in Figure 2c was chosen. Note the inclusion of a 3x3 Average pool layer for the skip connection with filter concatenation instead of elementwise addition to increase channel dimension.



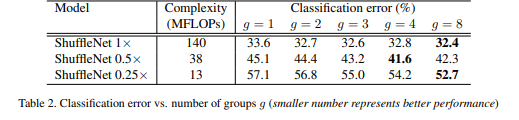
The ResNeXt setup (Figure 2a) which has *c*\**h*\**w* input and *m* bottleneck channels requires *hw(2cm+9m^2)* FLOPS while the ShuffleNet unit requires *hw(2cm/g+9m)* FLOPS where g is the number of groups for convolutions. As you can see, ShuffleNet can utilize wider feature maps which substantially less computational power. On low powered devices, they found that depth-wise convolution is difficult to implement efficiently while maintaining decent performance.

Now for the moment we’ve been waiting for—the actual structure of ShuffleNet (Table 1 below). We see that the initial stride in each stage is set equal to two with all other parameters held constant. Looking at the right side of the table at the output channels, each subsequent *stage* has its output channels doubled to widen the feature maps. The number of bottleneck channels to use was determined by prior research (*Deep residual learning for image recognition)* and set to ¼ of the output channels for a ShuffleNet unit*.* The hyperparameter *g* controls the number of output channels which, as seen below, has little effect on overall computation cost. To control for complexity, they apply a scale factor *s* on the number of channels.



Next we will look at how they evaluated the model. They chose to use ImageNet 2012 and top-1 performance on the ImageNet validation set as their primary means of evaluating performance. They looked at the inclusion/exclusion of both pointwise group convolutions and channel shuffle operations. Group numbers tested ranged from 1 through 8 where 1 is equivalent to an Xception like structure. Along with these different group sizes, they also tested different complexities. The resulting network that had the least classification error was *g*=8 for ShuffleNet 1x (original size of the network). In general, the smaller the model is the more it benefits from additional groups (going across the table). The researchers hypothesize that since more group convolutions means more feature map channels, this equals more complexity and thus better performance.

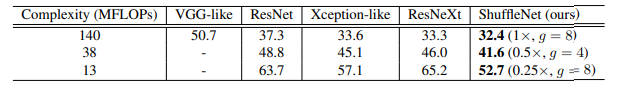
The total results can be seen below in Table 2



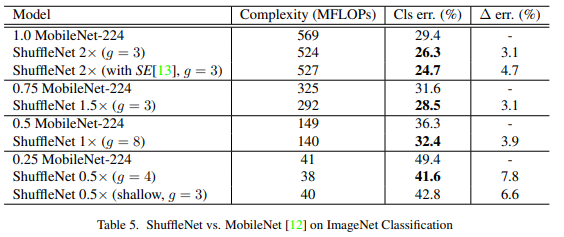
Next they evaluated channel shuffle vs no shuffle. In all circumstances testing, channel shuffling increased model performance which is especially true for large group numbers. They test the effects of shuffling versus other structures such as those found in VGG, ResNet, GoogleNet, ResNeXt, and Xception by replacing shuffling layers with layers found in the networks above. The table below describes the changes made:

|  |  |
| --- | --- |
| **Network** | **Change made to channel shuffle** |
| VGG-like | Two 3x3 convolution layers with the addition of batch norm after each convolution. |
| ResNet | Bottleneck design as pictured in figure 2. |
| Xception-like | Removal of pointwise group convolution and shuffle. Inclusion of depthwise separable convolution i.e. an Xception-like structure. |
| ResNeXt | Tuning the cardinality and bottleneck ratio (the number of bottleneck channels to output channels) to 16 and 1:2, respectively. |

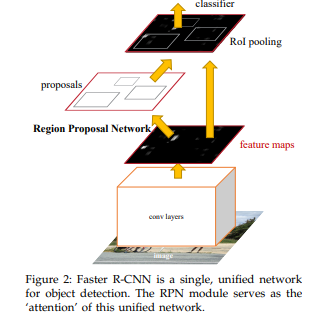
Given these changes, the results of model evaluation are as follows:



It is extremely surprising that the researchers were able to beat many of the leading networks in image classification given the same complexity! It is important to remember that no inception series were implemented because these structures are too complex and would not have been able to be implemented in a small network, so they were removed for a better apples-to-apples comparison. Something I think would be interesting is to compare a CNN setup with a few layers that could be run on a home GPU and compare performance with and without channel shuffle layer. Next, the paper explores how their net compares to a similar network that is efficient for mobile devices called *MobileNets*. MobileNet primarily utilizes depthwise separable convolutions to achieve great results. The table below demonstrates that ShuffleNet performs considerably better than several MobileNet architectures at different levels of complexity. It is truly remarkable for the least complex models that ShuffleNet has an advantage over MobileNet of nearly 8%! Additionally, ShuffleNet obtained similar performance as AlexNet while being theoretically **18 times** faster.



To test for generalization of the network, they used MS COCO object detection and a Faster-RCNN network architecture referenced in *Faster r-cnn: Towards real time object detection with region proposal networks* which is pictured below. They found that at similar complexity levels as measured in MFLOPS, ShuffleNet surpassed MobileNet in COCO detection scores. They assert that the superior performance is somewhat due to ShuffleNet’s simplicity and lack of “bells and whistles.”



Lastly, they evaluate performance on an ARM platform. Counterintuitive to pervious results, they found that ShuffleNets with larger group numbers performed worse. They find that a parameter of *g=3* to have a balance of accuracy and inference time. While the theoretical speedup compared to AlexNet was 18 times, they found the actual speedup was an impressive 13 times faster under comparable classification accuracy.

After reading this paper, it is truly impressive that a relatively simple architecture and fast model can achieve some impressive results. Even though they compared their network to others with some important features removed it was still interesting to see better performance given the equivalent computational constraints. While the main implications for the paper were impacts on mobile processing machines I can potentially see its use for an enthusiast at home or even at enterprise scale where resources may still not be quite abundant.

**Paper 2: Learning to Compare Image Patches via Convolutional Neural Networks**

**Sources:** [**https://openaccess.thecvf.com/content\_cvpr\_2015/papers/Zagoruyko\_Learning\_to\_Compare\_2015\_CVPR\_paper.pdf**](https://openaccess.thecvf.com/content_cvpr_2015/papers/Zagoruyko_Learning_to_Compare_2015_CVPR_paper.pdf)

[**https://opencv-python-tutroals.readthedocs.io/en/latest/py\_tutorials/py\_calib3d/py\_epipolar\_geometry/py\_epipolar\_geometry.html**](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_epipolar_geometry/py_epipolar_geometry.html)

This paper examines how to use a CNN-based model that is designed to account for changes in image appearance (or similarity). I have attempted using other non-deep learning techniques such as SIFT, SURF, and ORB in the past to try to match moon images to match features but was not very successful. This paper interested me because it could potentially provide a way to solve such a task and an opportunity for a large scale project.

Computer vision tasks have often involved tasks similar to the one stated above, as well as stitching together images, object recognition, classification, etc. There are many issues when it comes to comparing whether two images are similar as there are a variety of factors that affect the final appearance of an image. For example, viewpoint, lighting/shading, differences in camera settings themselves are among just a few factors that can influence the final image. The paper mentions that there are many shortcomings to previous algorithms such as SIFT that suffer from manually designed descriptors.

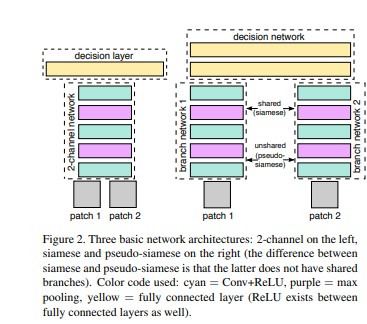
The goal of the paper is to generate a patch similarity function from scratch without using manually designed features. Instead they want to use a method (i.e. deep learning) to learn this function through pairs of image patches. The main contributions are outlined as follows:

1. Learning directly from image data rather than pre-defined features
2. Explore and propose a variety of network architectures for completing the desired task
3. Benchmark the performance on multiple datasets and demonstrate superiority over previous methods such as SIFT and DAISY

Previously, there had been one CNN-based solution to comparing image patches as notes in the paper *Computing the stereo matching cost with a convolutional neural network.* The model setup and architecture only looked at very small patches, but did perform well on the KITTI dataset, a stereo imaging dataset. The input to the proposed network would consist of a pair of image patches with no limitations on the number of channels. They do note that in order to compare to other models, they must use grayscale patches but performance was increased when using color patches for their network. Input images are low resolution as they are only of size 64x64. They do not state whether this choice was due to limitations in computation/training or out of necessity of what is available. Three models were explored in which information sharing can take place—2-channel, Siamese, and Pseudo-siamese.

Next, the model architectures are explained in better detail. The Siamese network uses two branches that share the same architecture and weights. Each branch takes one image of the pair and applies the “traditional” convolutional operations such as convolutions, pooling, and activations. The outputs of each branch are concatenated and fed to a top network that has fully connected layers. Branches are considered descriptor modules and the top network is the similarity function.

Pseudo-siamese networks are considered a hybrid between Siamese and 2-channel networks. The structure is similar to that of a Siamese network but the weights of the branches are not shared. The parameters under this framework are increased and therefore training is more difficult, but evaluation is much faster. 2-channel networks completely remove the idea of a “descriptor” in the model. Instead, pairs of images are considered a 2 channel image. The bottom part of the network is the convolutions/relu/maxpool while the top is a fully connected linear decision layer with a single output. The researchers state that this approach provides the most flexibility and is fast to train but at a cost of slower performance at testing time as it suffers from an exploding number of combinations being tested against each other. Please see the figure below for a visual representation of these three networks.



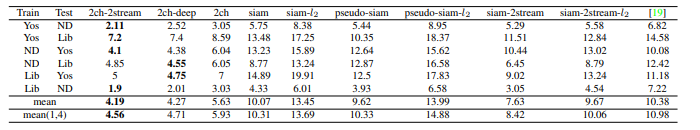
Other networks were explored such as a deep network (ConvNet) with some additional modifications, a central-surround two-stream network, and a spatial pyramid pooling (SPP) network. We have already explored ConvNet in early in our lectures for CNNs, so I will not elaborate here. The other two methods are as follows:

* Central surround two-stream networks contain two streams which allow for processing images in two different resolutions. Two 32x32 patches are sent through the high resolution stream that are made by cropping the center 64x64 patch of an image. The low resolution steam receives identical patch sized which are made from downsampling at half the original pair of input patches. These can then be used in the networks described in the prior paragraphs. The benefit to this approach is that it incorporates multiple resolutions which improves model performance for image matching. They provide the objective function to minimize as

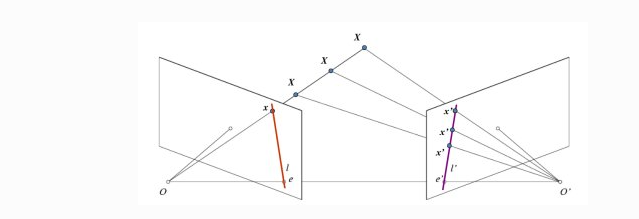
Where *w* denotes weights, *oi* is the output for the i-th sample, and yi is {-1,1} (a match or non match)

* The SPP network is unique in that it allows image patches of arbitrary size to be compared where this is not possible with other networks as the dimensions must be pre-defined. The SPP networks includes a framework similar to the SIFT algorithm that utilizes a spatial pyramid pooling layer between the convolutional and fully connected layers. This layer aggregates features of the last convolutional layer through spatial pooling where the size of pooling is dependent on the input size. The researchers propose including such a structure in all the aforementioned networks.

Next, we move on to the learning and evaluation section of the paper. For all models, they used a hinge-based loss term with l2 regularization. The training and testing datasets consisted of Yosemite, Notre Dame, and Liberty which are image patch datasets. The main conclusion to be drawn from the results below is that the 2-channel networks denoted with “2ch” outperform all other methods by a substantial margin. The researchers state that this implies there is importance in jointly using the information in both image patches from the beginning of the network. It is interesting to note, though, that all of these methods outperformed the current methods for comparing image patches and SIFT.

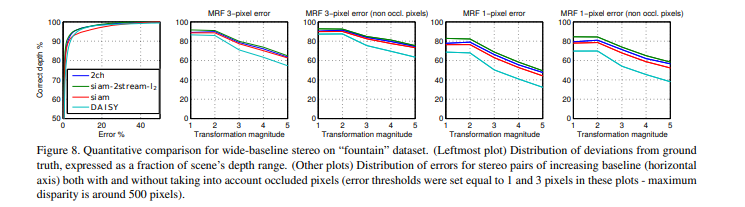


For their evaluation of stereo imaging they used the dataset used in *On Benchmarking Camera Calibration and Multi-View Stereo for High Resolution Imagery* and chose to use the algorithm DAISY for comparison, an image descriptor that is based on gradient orientation histograms such as those found in SIFT. While going through this section, I found it somewhat difficult to follow on first glance and required further research to understand more about computer vision in the scope of stereo imaging. The OpenCV python library had a great introduction on the basic concepts needed to understand this section and the link can be found in my sources.

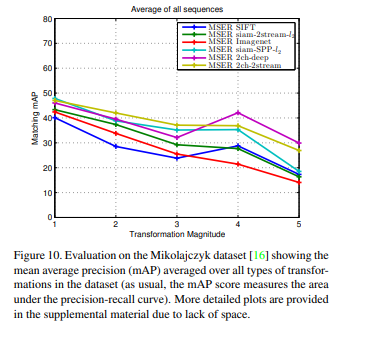


With stereo imaging, we use two cameras such that we can triangulate the location of a 3D point. We will use the above image to follow as an example. The left pane represents the projection of a point with one camera in which we are unable to determine its depth. Using two cameras we are able to obtain a line on the right pane consisting of points *x’* that allows for the triangulation. This new line is called the epiline in which we can find the point x by searching along the epiline. According to OpenCV, all points have corresponding epilines in the other image on what is called the epipolar plane (**XOO’** in the image above). An epipole is the point of intersection of a line through camera centers and the image planes, denoted epsilon or epsilon prime in the image.

With this vocabulary we can move forward in the paper. They took patches one patch from the left image and all patches from the corresponding epipolar line from the right image to obtain multiple batches of images. They then develop cost functions to be used for model training with Siamese and pseudo Siamese networks. Additionally, they note considerations for speeding up training like treating fully connected layers as 1x1 convolutions as noted in prior research. Upon evaluation it was found that 2-channel architectures produced much better and detailed depth maps than DAISY and that Siamese networks performed much better than DAISY across multiple error thresholds. Results can be seen below:



For testing local descriptors they tested their networks on the Mikolajczyk dataset for local descriptors evaluation. This dataset has 48 images which change viewpoints, blur, compression, lighting, and zoom. They applied detectors to both pairs of images to extract keypoints in which they use MSER (maximally stable extremal regions) to extract patches from input images. Depending on the network, outputs or patch-pairs were extracted to assign a score. These scores are summarized in the chart below:



In conclusion, the networks provided in the paper significantly outperformed SIFT when it came to matching image pairs. The 2-channel network showed superiority—beating out SIFT as well as other network architectures. One of the lasting points the paper makes is that a larger training set would likely improve performance of their approach even more as at the time of writing the paper the training set was quite small.