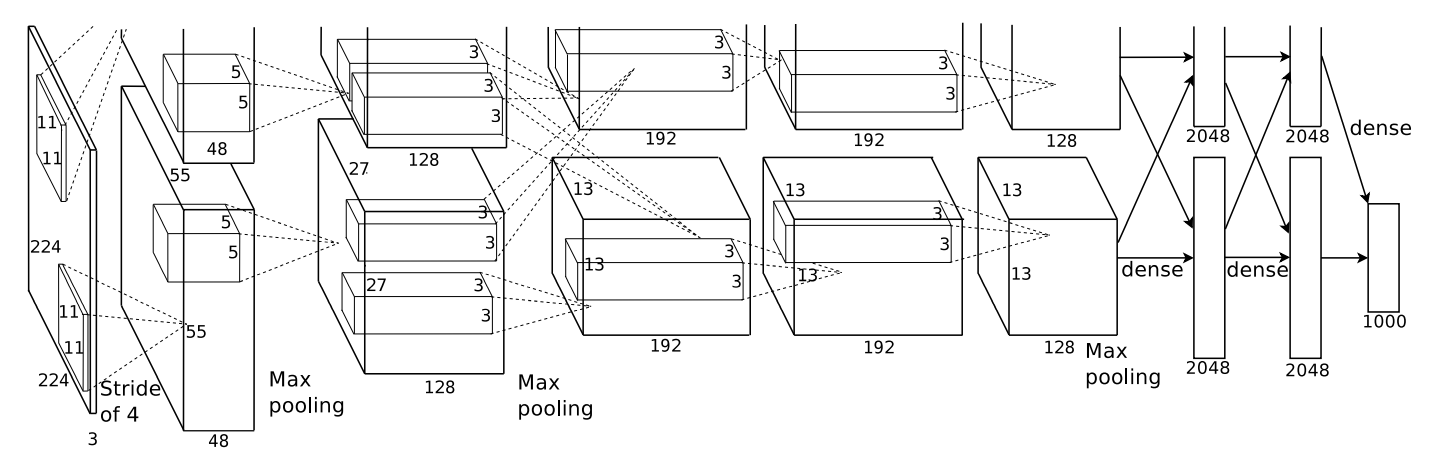
**ImageNet Classification with Deep Convolutional Neural Networks**

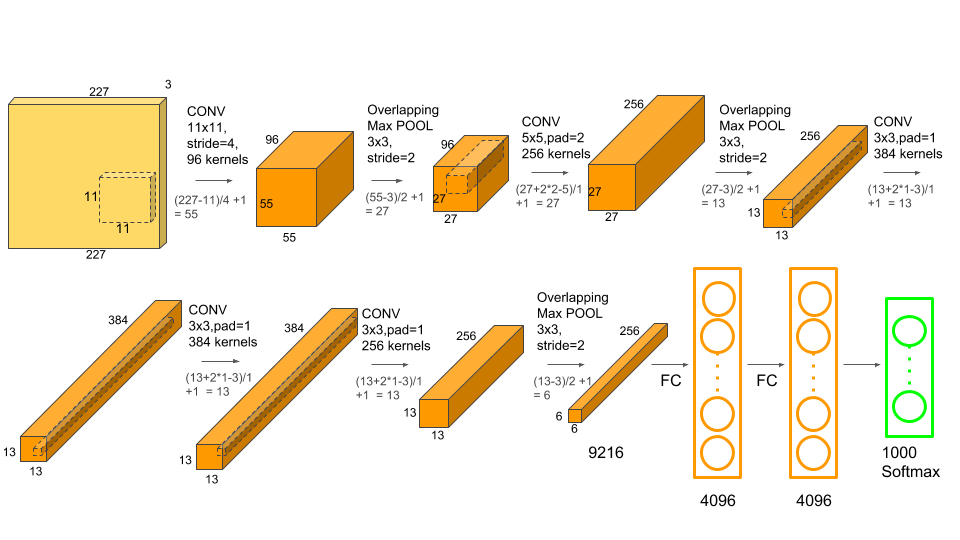
*Alex Krizhevsky Ilya Sutskever Geoffrey E. Hinton*

* Simple object classification problems can be solved with small datasets, but for realistic object detection, we need huge data.
* CNN’s design makes it possible to exploit the locality of pixels properties in images. Thus, CNN work better compared to MLP for CV tasks.
* ImageNet has over 15 million images and 22k categories
* ILSVRC dataset has 1.2 million training images, 50k validation images, and 150k testing images from 1k categories
* The competition uses top-1 and top-5 errors for comparison
* Images are of different scales, so rescaled to 256\*256
* Mean is subtracted from each pixel
* The network has 5 conv. layers and 3 FC layers.
* Used ReLU because it is non-saturating, it trains faster compared to tanh/sigmoid, and it doesn’t require input normalization.
* Training was split on two GPUs, with **most** of the layers connected to the kernels from the previous layers on the same GPU.
* Uses local response normalization *(proven to be not effective in VGG network paper)*
* Uses max pooling layers with stride 2 and kernel size 3\*3. Since stride is smaller than the filter size, kernels overlap.
* Softmax is applied to the last layer output to get 1000 probabilities.
* Maximizes multinomial logistic regression objective
* ReLU after each conv/FC layer
* 
* Input size: (227, 227, 3)

Even though the above image shows (224, 224) as input dimensions, it seems it should be (227, 227) because after applying the first conv. layer, we expect the output shape to be (55, 55)

(<https://learnopencv.com/understanding-alexnet/>)

So, we have images of size (256, 256) from which (227, 227) crops are taken out and used for training.



* First conv. layer: 96 filters of 11\*11\*3 dimensions with stride 4
* Local Response normalization
* Maxpooling: pool size 3\*3 with stride of 2
* Second conv. layer: 256 filters of 5\*5\*48 dimensions with stride 1 and padding 2

(5\*5\*96 if you combine the computations of both the GPUs)

* Local Response normalization
* Maxpooling: pool size 3\*3 with stride of 2
* Third conv. layer: 384 filters of 3\*3\*128 dimensions with stride 1 and padding 1

(3\*3\*256 if you combine the computations of both the GPUs)

* Fourth conv. layer: 384 filters of 3\*3\*192 dimensions with stride 1 and padding 1

(3\*3\*384 if you combine the computations of both the GPUs)

* Fifth conv. layer: 256 filters of 3\*3\*192 dimensions with stride 1 and padding 1

(3\*3\*384 if you combine the computations of both the GPUs)

* Maxpooling: pool size 3\*3 with stride of 2
* FC layer: 4096 neurons
* Dropout: 0.5 probability
* FC layer: 4096 neurons
* Dropout: 0.5 probability
* FC layer: 1000 neurons
* Softmax
* The model has about 60 million parameters and 650k neurons
* Data augmentation:

To reduce overfitting data augmentation is necessary.

During training, take patches of size (227, 227) from the input images of size (256, 256); also include mirror reflections of all the patches

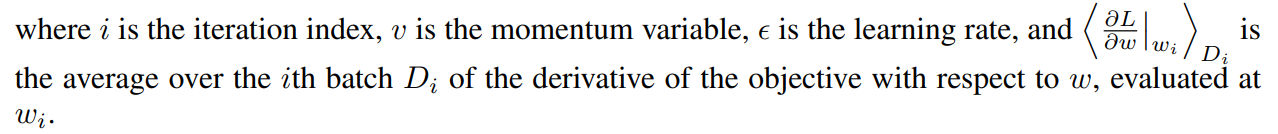
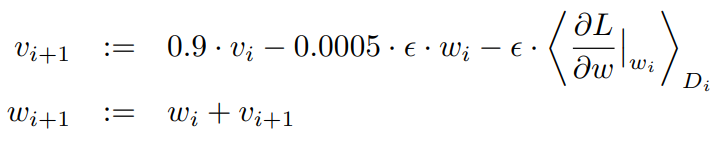
*(paper mentions (224, 224), which looks like a typo)*

During testing, take 5 patches from the input image (the four corners and the central patch) and their reflections. Take average of the predictions on these 10 patches to get the final output.

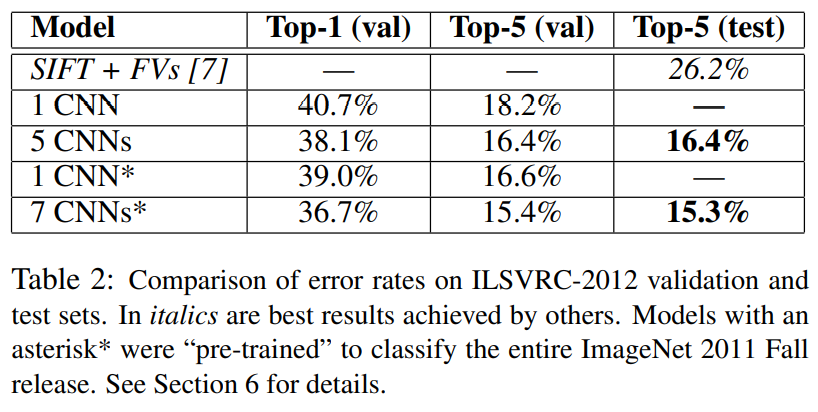
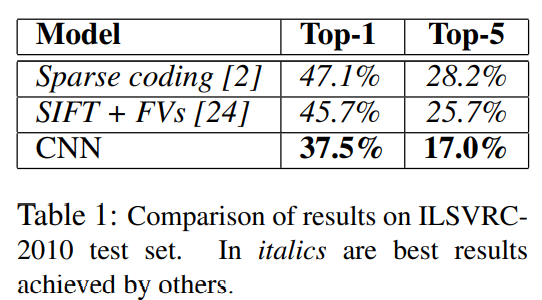
Another method of augmentation is to alter the pixel intensities by applying PCA on the input images, and then adding below to image pixels

Each represents an Eigen vector, each represents corresponding Eigen value, and is some random number from Gaussian distribution with mean 0 and std 0.1

* As mentioned before, dropout is used to deal with overfitting.
* Used SGD with a batch size of 128 examples, momentum of 0.9, and weight decay of 0.0005.
* Update rule



* Parameter Initialization:
  + All weights: Gaussian distribution with mean 0 and std 0.01
  + Biases in 2nd, 4th, and 5th conv layers and all FC layers: 1
  + Biases in remaining layers: 0
* Initial learning rate: 0.01
* New learning rate = learning rate / 10, whenever validation error stopped improving
* Epochs: 90
* **Results:**



* Find AlexNet & VGG weights here: <http://www.cs.toronto.edu/~guerzhoy/tf_alexnet/>