1 Theory (60 points)

1. [Deep Architectures] A recursive/inductive translation of a boolean circuit into an equisatisfiable boolean formula works as follows:
   
   - Start with the gate that produces the circuit output and inductively express each of the gate’s inputs as formulas.
   - Obtain the formula for the circuit by writing an expression that applies the gate’s function to its inputs’ formulas.

   A boolean formula is equisatisfiable with a boolean circuit if input configurations that make the boolean circuit output 1 are in a one-to-one correspondence with variable configurations that satisfy the formula. Equivalently, the circuit is satisfiable if and only if the formula is satisfiable. An example of applying the recursive procedure is shown in the figure below.

   ![Example Circuit](image)

   Describe a circuit of size $n$ that, when converted to an equisatisfiable formula by the method above, yields a formula whose size is exponential in $n$.

2. [Deep Architectures] Design a different translation method that creates an equisatisfiable boolean formula whose size is linear in the size of the boolean circuit.

3. [CNN Complexity] Consider the final convolutional network from step 7 in the implementation part below. What is the total number of parameters? Show your work, i.e., a breakdown of the total number of parameters, layer by layer. Describe at least three different ways of reducing the number of parameters in this network, and discuss their relative advantages/disadvantages.
2 Implementation (100 points)

Download the skeleton code from http://ace.cs.ohio.edu/~razvan/courses/dl6890/hw/hw05.zip. Implement the ConvNet for digit classification, as explained in the 7 steps below, using TENSORFLOW. Make sure that you organize your code in folders as shown in the table below.

```
dl6890/
    hw05/
        tensorflow/
            train_cnn.py
            cnnExercise.py
            expand_mnist.py
            output.txt
        mnist/
            data/
```

Write code only in the files indicated in bold. The traces from running `python3 cnnExercise.py <mode>` for each of the 7 steps below should be saved in `output.txt`. In each step you will train the corresponding architecture and report the accuracy on the test data. Hyperparameters are kept unchanged from one step to the next, unless changes are explicitly mentioned. If your TensorFlow installation does not use GPUs, set `has_GPU = False`. It is recommended that you read and understand the code in `cnnExercise.py` and `expand_mnist.py`. Most of the functionality needed for completing this assignment is implemented in the package `tf.layers`.

**Step 1: Baseline with one hidden layer**: Create a shallow architecture using a single hidden layer, fully connected, containing 100 neurons, with sigmoid activation function. Train for 60 epochs, using SGD with a learning rate of 0.1, a mini-batch size of 10, and no regularization.

**Step 2: One convolutional + one hidden layer**: Insert a convolutional layer at the beginning of the network, followed by a max-pooling layer and the fully connected layer from step 1. Use 5x5 local receptive fields, a stride of 1, and 20 kernels. The max-pooling layer should combine features using 2x2 pooling windows. The overall architecture should look as in the figure below.

**Step 3: Two convolutional + one hidden layer**: Insert a second convolutional-pooling layer between the existing convolutional-pooling layer and the fully-connected hidden layer. Use a 5x5 local receptive field for 40 kernels and pool over 2x2 regions.

**Step 4: Two convolutional + one hidden layer, ReLU**: Replace `sigmoid` with `ReLU` as activation function in the entire network from step 3. Train the model using a new learning rate of 0.03.

**Step 5: Two convolutional + one hidden layer, ReLU, data augmentation**: Augment the training data by displacing each training image by a single pixel: up one pixel, down
one pixel, left one pixel, or right one pixel. Do this by running the program `expand_mnist.py` in the `mnist/data/` folder. Retrain the model from step 4 on the augmented dataset, this time using L2 regularization on the parameters of the FC layer, with a decay parameter of 0.1.

**Step 6: Two convolutional + two hidden layers, ReLU, data augmentation:** Add a second fully-connected layer with 100 neurons to the model from step 5. Use L2 regularization on the parameters of the two FC layers.

**Step 7: Two convolutional + two large hidden layers, ReLU, data augmentation, Dropout:** Increase the number of neurons on the fully connected layers to 1000. Do regularization by applying Dropout on the activations of the two fully-connected layers, using a dropout rate of 0.5. Train using 40 epochs.

### 3 Bonus (150 points)

Implement all the above using SciPy.

### 4 Submission

Turn in a hard copy of your homework report at the beginning of class on the due date. Electronically submit on Blackboard a hw05.zip file that contains the hw05 folder in which you change code **only in the required files**. The screen output produced when running the `cnnExercise.py` code should be redirected to (saved into) the `output.txt` file.

On a Linux system, creating the archive can be done using the command:

```
> zip -r hw05.zip hw05
```

Please observe the following when handing in homework:

1. Structure, indent, and format your code well.
2. Use adequate comments, both block and in-line to document your code.
3. On the theory assignment, clear and complete explanations and proofs of your results are as important as getting the right answer.
4. Make sure your code runs correctly when used in the directory structure shown above.