**IMPLEMENTATION**

**MODULES:**

* Dataset
* Importing the necessary libraries
* Retrieving the images
* Splitting the dataset
* Building the model
* Apply the model and plot the graphs for accuracy and loss
* Accuracy on test set
* Saving the Trained Model

**MODULES DESCSRIPTION:**

**Dataset:**

In the first module, we developed the system to get the input dataset for the training and testing purpose. We given the data set in model folder.

The dataset consists of 2222 Diabetic Retinopathy images.

**Importing the necessary libraries:**

We will be using Python language for this. First we will import the necessary libraries such as keras for building the main model, sklearn for splitting the training and test data, PIL for converting the images into array of numbers and other libraries such as pandas, numpy, matplotlib and tensorflow.

**Retrieving the images:**

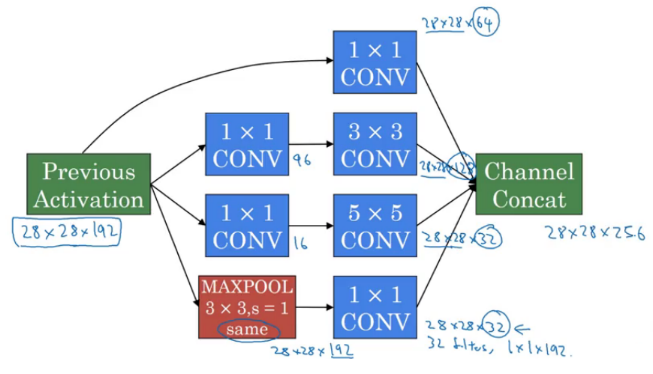
We will retrieve the images and their labels. Then resize the images to (224,224) as all images should have same size for recognition. Then convert the images into numpy array.

**Splitting the dataset:**

Split the dataset into train and test. 80% train data and 20% test data.

# Inception Networks

This is how an inception block looks:



# Inception v3

The Premise

The authors noted that the auxiliary classifiers didn’t contribute much until near the end of the training process, when accuracies were nearing saturation. They argued that they function as regularizes, especially if they have BatchNorm or Dropout operations.

Possibilities to improve on the Inception v2 without drastically changing the modules were to be investigated.

# The Solution

1. **Inception Net v3** incorporated all of the above upgrades stated for Inception v2, and in addition used the following:
2. RMSProp Optimizer.
3. Factorized 7x7 convolutions.
4. BatchNorm in the Auxillary Classifiers.
5. Label Smoothing (A type of regularizing component added to the loss formula that prevents the network from becoming too confident about a class. Prevents over fitting).

**Building the model:**

The concept of convolutional neural networks. They are very successful in image recognition. The key part to understand, which distinguishes CNN from traditional neural networks, is the **convolution** operation. Having an image at the input, CNN scans it many times to look for certain **features**. This scanning (convolution) can be set with 2 main parameters: stride and padding type. As we see on below picture, process of the first convolution gives us a set of new frames, shown here in the second column (layer). Each frame contains an information about one feature and its presence in scanned image. Resulting frame will have larger values in places where a feature is strongly visible and lower values where there are no or little such features. Afterwards, the process is repeated for each of obtained frames for a chosen number of times. In this project I chose a classic InceptionV3 model which contains only two convolution layers.

The latter layer we are convolving, the more high-level features are being searched. It works similarly to human perception. To give an example, below is a very descriptive picture with features which are searched on different CNN layers. As you can see, the application of this model is face recognition. You may ask how the model knows which features to seek. If you construct the CNN from the beginning, searched features are random. Then, during training process, weights between neurons are being adjusted and slowly CNN starts to find such features which enable to meet predefined goal, i.e. to recognize successfully images from the training set.

Between described layers there are also **pooling** (sub-sampling) operations which reduce dimensions of resulted frames. Furthermore, after each convolution we apply a non-linear function (called **ReLU**) to the resulted frame to introduce non-linearity to the model.

Eventually, there are also fully connected layers at the end of the network. The last set of frames obtained from convolution operations is flattened to get a one-dimensional vector of neurons. From this point we put a standard, fully-connected neural network. At the very end, for classification problems, there is a softmax layer. It transforms results of the model to probabilities of a correct guess of five class

**Apply the model and plot the graphs for accuracy and loss:**

We will compile the model and apply it using fit function. The batch size will be 10. Then we will plot the graphs for accuracy and loss. We got average validation accuracy of 95.6% and average training accuracy of 97.3%.

**Accuracy on test set:**

We got an accuracy of 97.3% on test set.

**Saving the Trained Model:**

Once you’re confident enough to take your trained and tested model into the production-ready environment, the first step is to save it into a .h5 or . pkl file using a library like pickle .

Make sure you have pickle installed in your environment.

Next, let’s import the module and dump the model into .h5 file