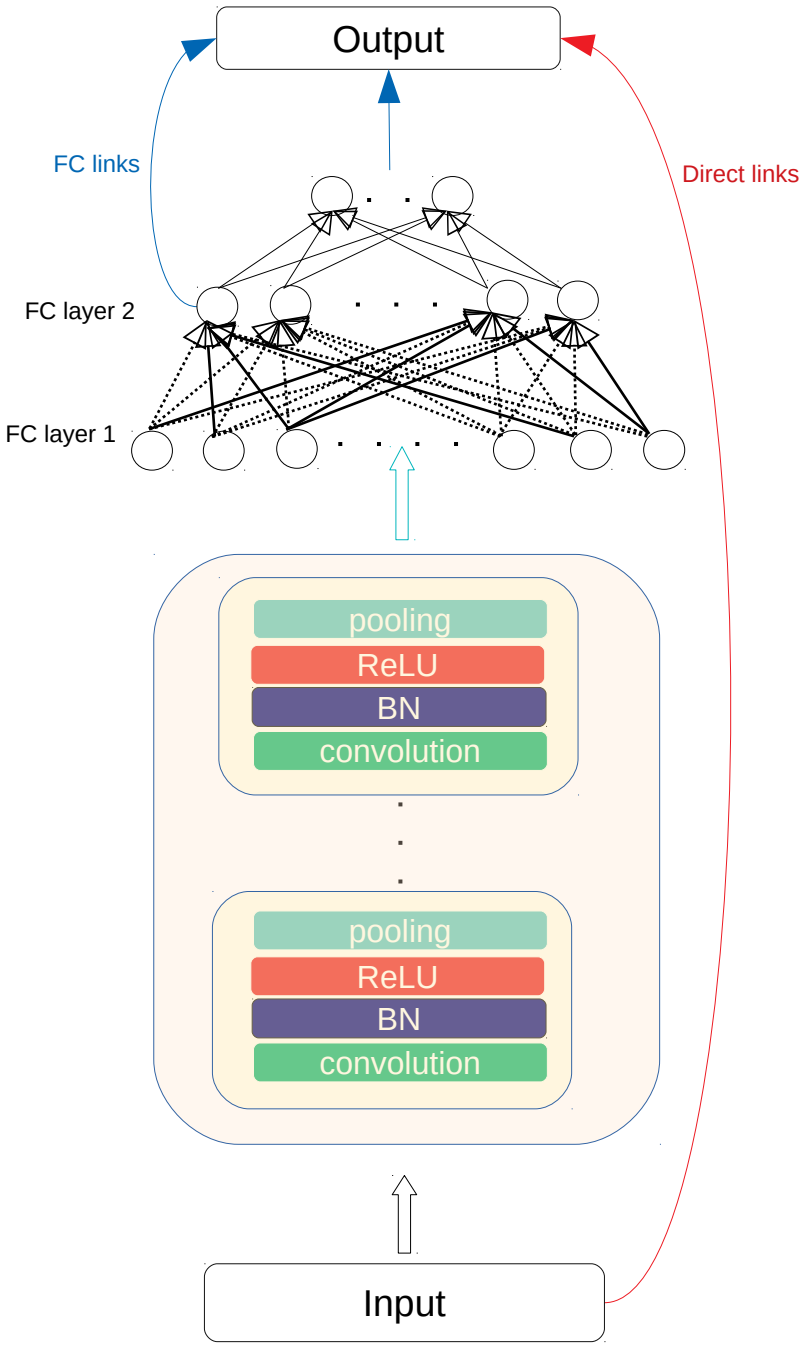

Figure 1 Framework of proposed convolutional RVFL network. It consists of several convolutional layers stacked on top of each other whose parameters are randomly generated and kept fixed during the training. Following a fully-connected layer linked with directly output layer with dropout probability of 0.5 whose parameters are randomly generated and kept fixed during the training. At the end, another fully-connected layer with softmax transfer function whose parameters are analytically computed via pseudoinerse learning. Output layer is fed with original input data and output of designed fully-connected layers.

```
function net= cdRVFLtrain(input, target, numberofconvlayer, fclayerstructure)
% cdRVFLtrain: ConvNET Random Vector Functional Link training function
% 0.5 probality rate of dropout
%Output Parameters
%         net: structure that includes network parameters.
%         fcweights, stride, convkernelflipped, poolkernel
%         outputlayerweights, fclayerstructure, numberofconvlayer
%
%Input Parameters
%         input: input data (each row represent different observations)
%         target: desired outputs
%         numberofconvlayer: conv layer neuron numbers
%         fclayerstructure: neuron numbers of fully connected layers, for instance [5 8]
%
% Example Usage
%         input=rand(3,25);
%         target=rand(3,1);
%         net=cdRVFLtrain(input, target, 5, [8,3])
%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %                              TRAIN                                    %
% %            ConvNET Random Vector Functional Link                      %
% %                                                                       %
% %                     Apdullah Yayik, 2019                              %
% %                     apdullahyayik@gmail.com                           %
% %                                                                       %
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if isequal(size(target,1), size(input,1))==0
    error('Error: input and target sizes dismatch')
else
    net.numberofconvlayer=numberofconvlayer;
    convlayerouts=input;
    net.conv.kernel=rand(1,3); net.conv.kernelflipped=rot90(net.conv.kernel,2);
    net.pool.kernel=ones(1,2)/2; net.pool.stride=1;
    net.dropoutlayers=randperm(fclayerstructure(1),fclayerstructure(1)*0.5);

    for p=1:net.numberofconvlayer-1
        convlayerouts=conv2(convlayerouts, net.conv.kernelflipped, 'same');% convolution
        convlayerouts=trans(batchN(convlayerouts), 'ReLU'); % Batch Normalization & ReLU
        temp = conv2(convlayerouts, net.pool.kernel, 'valid'); % pooling
        convlayerouts = temp(1:net.pool.stride:end, 1:net.pool.stride:end);
    end

    net.fcweights{1,1}=rand( size(convlayerouts, 2),fclayerstructure(1));
    fclayerouts1=batchN(convlayerouts*net.fcweights{1,1}); % FC, Batch Normalization &
softmax
    fclayerouts1(:,net.dropoutlayers)=[];% dropout layer
    net.fcweights{1,2}=rand( size(fclayerouts1, 2),fclayerstructure(2));
    fclayerouts2=trans(batchN(fclayerouts1*net.fcweights{1,2}), 'softmax'); % FC, Batch
Normalization & softmax
    D=[input, fclayerouts1, fclayerouts2];
    net.outputlayerweights=pinv(D)*target; % Pseudoinverse learning, svd
end
```

```matlab
function y=cdRVFLtest(input,net)
% cdRVFLtest: ConvNET Random Vector Functional Link testing function
%
%Output Parameters
%         y: actul output
%
%Input Parameters
%         net: structure that includes network parameters.
%         fcweights, stride, convkernelflipped, poolkernel
%         outputlayerweights, fclayerstructure, numberofconvlayer
%
% Example Usage
%         input=rand(3,25);
%         target=rand(3,1);
%         net=cdRVFLtrain(input, target, 5, [8,3])
%         y=cdRVFLtest(input, net)
%        % check target and y values
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %                              TEST                              %
% %           ConvNET Random Vector Functional Link                %
% %                          (Avaraging)                           %
% %                     Apdullah Yayik, 2019                       %
% %                     apdullahyayik@gmail.com                    %
% %                                                                %
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

convlayerouts=input;
for p=1:net.numberofconvlayer-1
    convlayerouts=conv2(convlayerouts, net.conv.kernelflipped, 'same'); % convolution
    convlayerouts=trans(batchN(convlayerouts), 'ReLU'); % Batch Normalization & ReLU
    temp = conv2(convlayerouts, net.pool.kernel, 'valid'); % pooling
    convlayerouts = temp(1:net.pool.stride:end, 1:net.pool.stride:end);
end

fclayerouts1=batchN(convlayerouts*net.fcweights{1,1}); % FC, Batch Normalization & softmax
fclayerouts1(:,net.dropoutlayers)=[]; % dropout layer
fclayerouts2=trans(batchN(fclayerouts1*net.fcweights{1,2}), 'softmax'); % FC, Batch
Normalization & softmax
D=[input, fclayerouts1,fclayerouts2];
y=D*net.outputlayerweights;
end



function n=batchN(H)
% batchN: Batch normalization
% Toffe and Szegedy, 2015
% recomended before non-linear transfering
%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %                                                                %
% %                     Batch Normalization                        %
% %                                                                %
% %                     Apdullah Yayik, 2019                       %
% %                     apdullahyayik@gmail.com                    %
% %                                                                %
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

n=(H-mean(H,2))./std(H,0,2);
end


function Y=trans(x, funct)
%AKTIVASYONFONK: Aktivasyon fonksiyonlar?n? ve türevlerini hesaplar.
% e?er durum:=1--> ileri besleme
```

```
% Y: katman ç?k??, x: giri? ve a??rl?k iç çarp?m?
% e?er durum:=0--> türev (yerel gradient)
% Y: yerel gradient, x: katman ç?k?? de?eri
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %                                                     %
% %                   AKTIVASYONFONK                    %
% %                   Apdullah Yay?k, 2016              %
% %                                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% switch durum
%     case 1
        switch funct
            case 'sigmoid'
                Y=sigm(x);
            case 'tangentH'
                Y=tanh(x);
            case 'tangentH_opt'
                Y=tanhopt(x);
            case 'ReLU'
                Y=relu(x);
            case 'softmax'
                Y=softmax(x);
            case 'linear'
                Y=x;
        end
%     case 0
%         switch aktivasyon
%             case 'sigmoid'
%                 Y=x.*(1-x);
%             case 'tangentH'
%                 Y=1-(x.^2);
%             case 'tangentH_opt'
%                 Y = 1.7159 * 2/3 * (1 - 1/(1.7159)^2 * x.^2); %LeChun,1998
%             case 'ReLU'
%                 Y =1. * (x > 0);
%             case 'linear'
%                 Y=1;
%         end
% end

function Y = sigm(x)
Y = 1./(1+exp(-x));

function Y=relu(x)
Y=x .* (x > 0);

function Y=tanhopt(x)
Y=1.7159*tanh(2/3.*x);

% tanh Matlab kütüphanede mevcut
% function Y=tanh(x)
% Y=(exp(x)-exp(-x))./(exp(x)+exp(-x));

function Y=softmax(x)
shiftx = x - max(x);
exps = exp(shiftx);
Y=(exps)./sum(exps);

% Y=(exp(x - max(x)))./sum(exp(x - max(x)));
```