

Εισαγωγή

Γενικότερα, τα νευρωνικά δίκτυα, γνωστά και ως τεχνητά νευρωνικά δίκτυα, αποτελούν υποσύνολο της μηχανικής μάθησης και βρίσκονται στην καρδιά των αλγορίθμων βαθιάς μάθησης. Το όνομα και η δομή τους είναι εμπνευσμένα από τον ανθρώπινο εγκέφαλο, μιμούμενοι τον τρόπο που λειτουργούν οι βιολογικοί νευρώνες.

Τα νευρωνικά δίκτυα βασίζονται σε δεδομένα εκπαίδευσης για να μάθουν και να βελτιώσουν την ακρίβειά τους με την πάροδο του χρόνου. Αξίζει να τονιστεί ότι αποτελούν ισχυρά εργαλεία στον τομέα της τεχνητής νοημοσύνης, επιτρέποντάς μας να ταξινομούμε και να ομαδοποιούμε δεδομένα με υψηλή ταχύτητα.

Στο μέρος Α της άσκησης χρησιμοποιούμε τις εφαρμογές GUI του Machine Learning Toolbox, nctool για Data clustering και ntstool για Time-series analysis. Μέσω των 2 εφαρμογών που αναφέραμε δημιουργούμε νευρωνικά δίκτυα και τα εκπαιδεύουμε με κατάλληλα σύνολα δεδομένων. Τέλος, λαμβάνει χώρα η αξιολόγηση των δικτύων.

Μεγάλη σύγχυση παρατηρείται στη χρήση των εννοιών Deep Learning και νευρωνικά δίκτυα. Ως αποτέλεσμα, αξίζει να σημειωθεί ότι το «βαθύ» στη βαθιά μάθηση αναφέρεται απλώς στο βάθος των στρωμάτων σε ένα νευρωνικό δίκτυο.(Μέρος Β της άσκησης: Εφαρμογή Deep Network Designer)

Τα νευρωνικά δίκτυα είναι επιπροσθέτως ιδιαίτερα για να βοηθήσουν τους ανθρώπους να λύσουν πολύπλοκα προβλήματα σε πραγματικές καταστάσεις. Μπορούν να μάθουν και να μοντελοποιήσουν τις σχέσεις μεταξύ εισόδων και εξόδων, να κάνουν γενικεύσεις, να αποκαλύπτουν σημαντικές πληροφορίες και να επιχειρούν διάφορες προβλέψεις. Υπάρχουν αρκετά παραδείγματα χρήσης νευρωνικών δικτύων, με στόχο τη λήψη απόφασης: Ανίχνευση απάτης με πιστωτική κάρτα, Ιατρική και διάγνωση ασθενειών, στοχευμένο μάρκετινγκ, πρόβλεψη ενεργειακής ζήτησης, αναγνώριση προσώπου και ομιλίας και πολλά άλλα.

Τα νευρωνικά δίκτυα οργανώνονται σε στρώματα. Το πρώτο στρώμα είναι η είσοδος, τα κρυφά στρώματα περιέχουν τους νευρώνες επεξεργασίας, και το τελευταίο είναι το στρώμα εξόδου. Οι νευρώνες ενός στρώματος συνδέονται με τους νευρώνες του επόμενου στρώματος, κάθε σύνδεση έχει ένα βάρος που ρυθμίζεται κατά την εκπαίδευση. Κάθε νευρώνας χρησιμοποιεί μια συνάρτηση ενεργοποίησης για να ρυθμίσει την έξοδό του. Η εκπαίδευση πραγματοποιείται σε επαναλαμβανόμενες εποχές με έναν καθορισμένο ρυθμό μάθησης που προσαρμόζεται κατάλληλα. Τα δεδομένα διαιρούνται σε σύνολα εκπαίδευσης, επικύρωσης και δοκιμής για να εκπαιδευτεί, να βελτιστοποιηθεί και να αξιολογηθεί το δίκτυο. Συνοπτικά, η εφαρμογή των παραπάνω μπορεί να διαφέρει ανάλογα με τον τύπο του προβλήματος.

Μέρος Α

- Δημιουργία, εκπαίδευση και αξιολόγηση νευρωνικών δικτύων μέσω της εφαρμογής nctool του Matlab(για Data Clustering)

Dataset: Iris Flowers

Αρχείο: nctool1.m

```
x = irisInputs;
z = irisTargets;

% Create a Self-Organizing Map
dimension1 = 10;
dimension2 = 10;
net = selforgmap([dimension1 dimension2]);

% Train the Network
[net,tr] = train(net,x);

% Test the Network
y = net(x);

% View the Network
view(net)

% Plots
%figure, plotsomtop(net)
%figure, plotsomnc(net)
%figure, plotsomnd(net)
%figure, plotsomplanes(net)
%figure, plotsomhits(net,x)
%figure, plotsompos(net,x)
```

Αρχείο: nctool2.m

```
function [y1] = myNeuralNetworkFunction(x1)

% ===== NEURAL NETWORK CONSTANTS =====

% Layer 1

% ===== SIMULATION =====

% Layer 1
z1 = negdist_apply(IW1_1,x1);
a1 = compet_apply(z1);
```

```

% Output 1
y1 = a1;
end

% ===== MODULE FUNCTIONS =====

% Negative Distance Weight Function
function z = negdist_apply(w,p,~)
[S,R] = size(w);
Q = size(p,2);
if isa(w,'gpuArray')
    z = iNegDistApplyGPU(w,p,R,S,Q);
else
    z = iNegDistApplyCPU(w,p,S,Q);
end
end

function z = iNegDistApplyCPU(w,p,S,Q)
z = zeros(S,Q);
if (Q<S)
    pt = p';
    for q=1:Q
        z(:,q) = sum(bsxfun(@minus,w,pt(q,:)).^2,2);
    end
else
    wt = w';
    for i=1:S
        z(i,:) = sum(bsxfun(@minus,wt(:,i),p).^2,1);
    end
end
z = -sqrt(z);
end

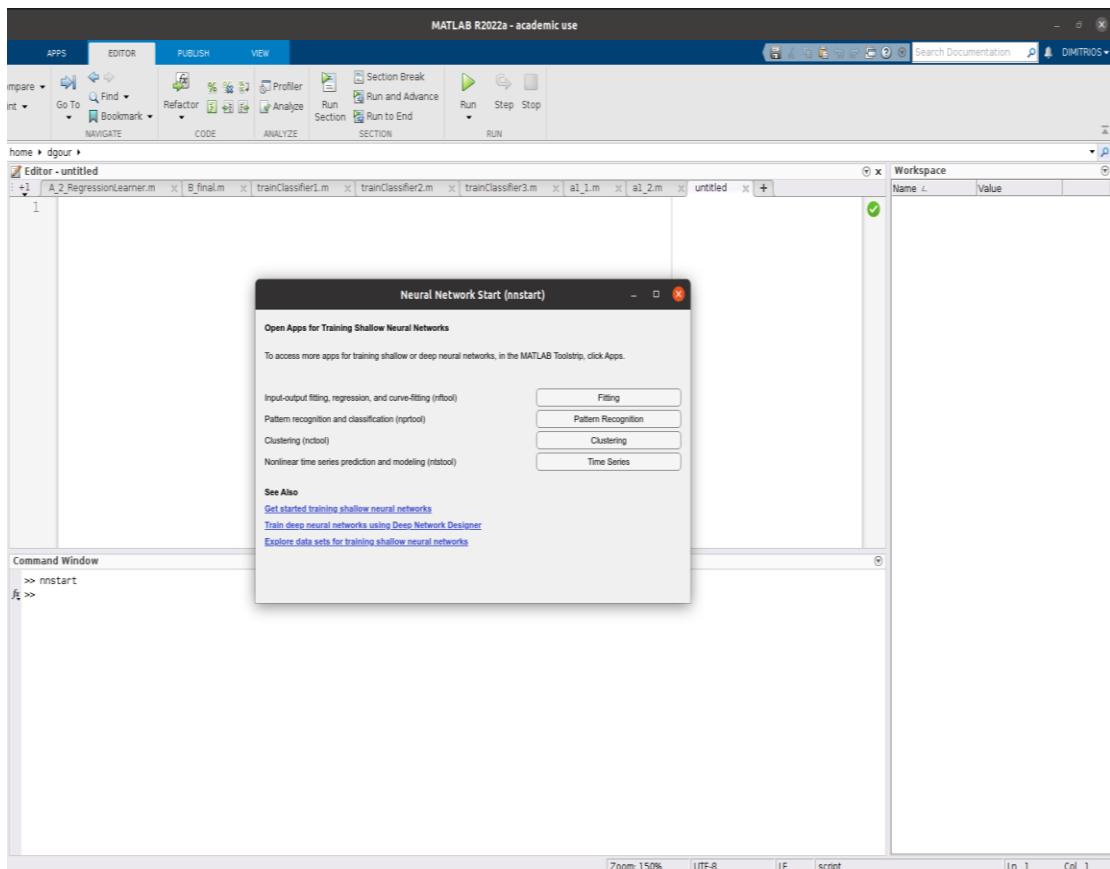
function z = iNegDistApplyGPU(w,p,R,S,Q)
p = reshape(p,1,R,Q);
sd = arrayfun(@iNegDistApplyGPUHelper,w,p);
z = -sqrt(reshape(sum(sd,2),S,Q));
end

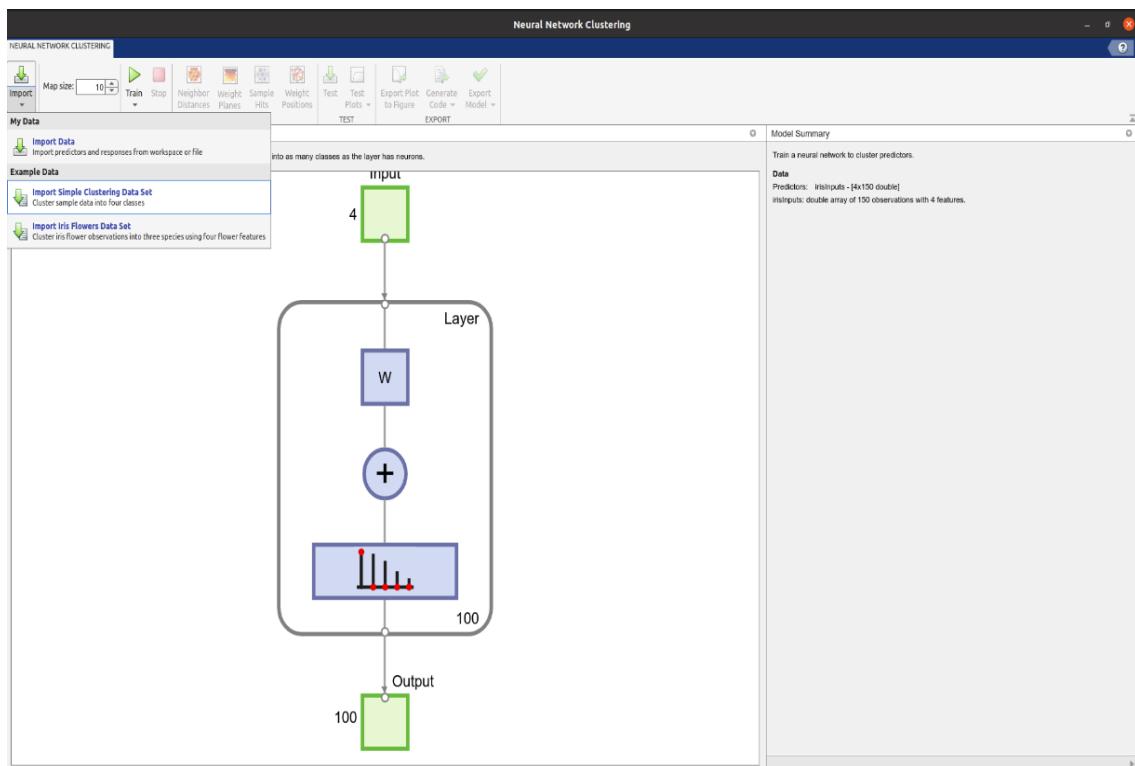
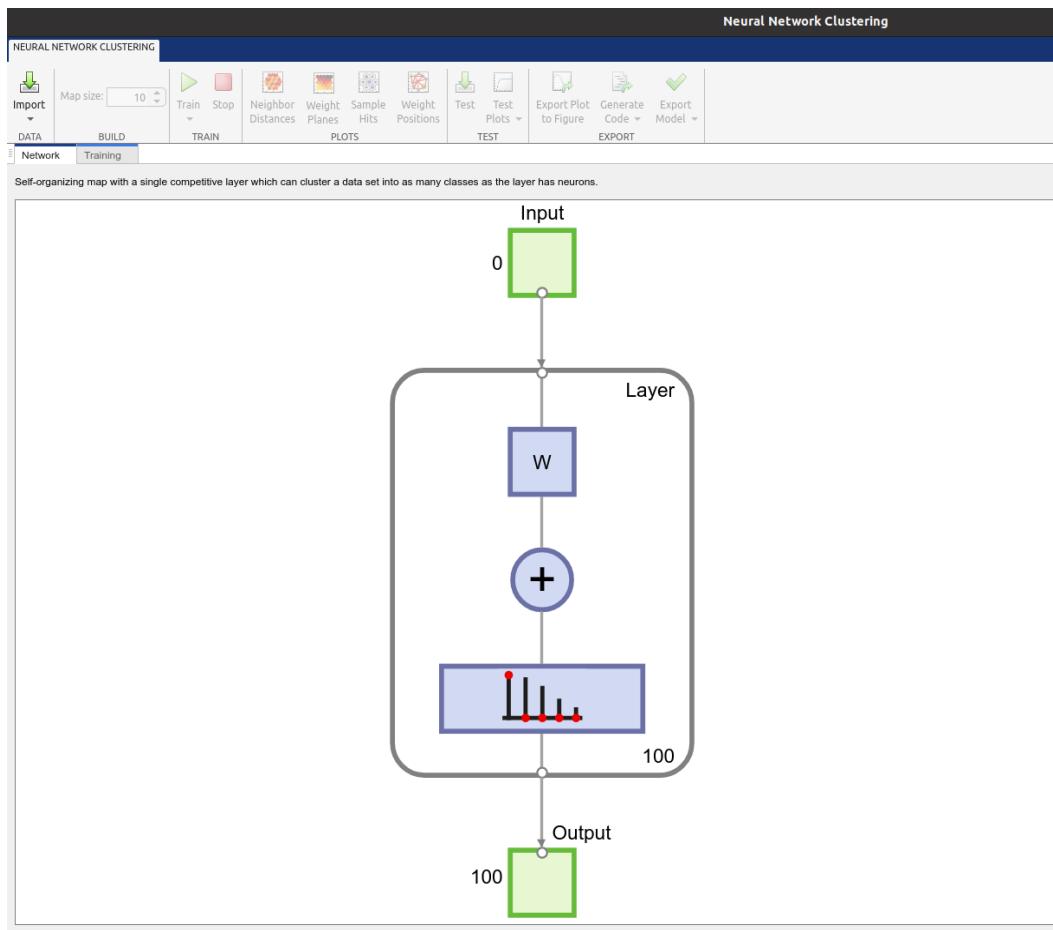
function sd = iNegDistApplyGPUHelper(w,p)
sd = (w-p).^.2;
end

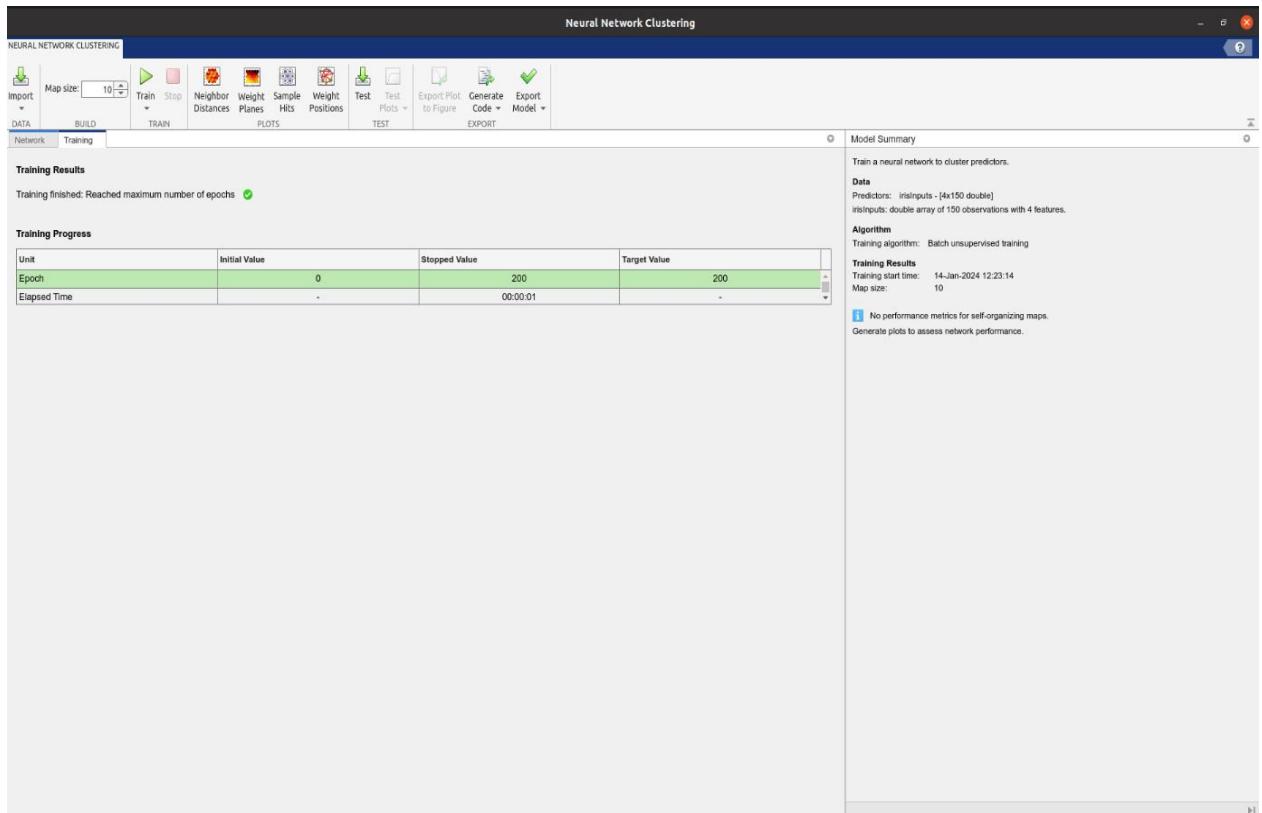
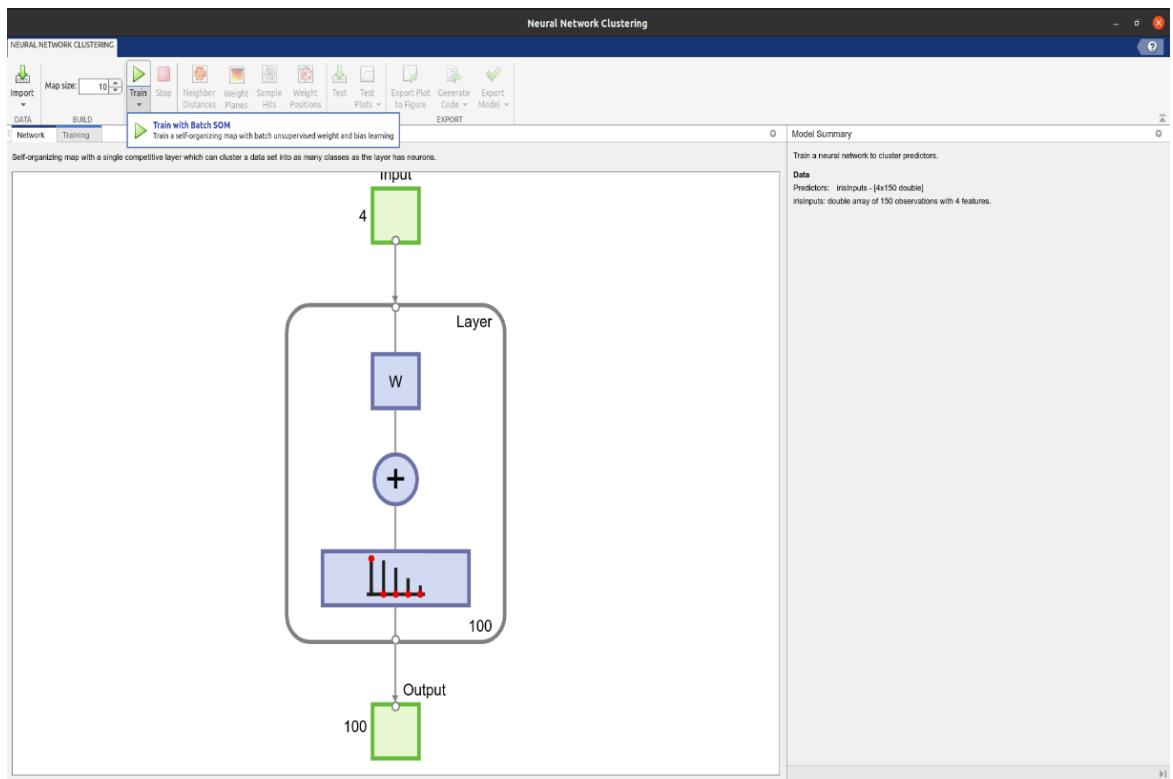
% Competitive Transfer Function
function a = compet_apply(n,~)
if isempty(n)
    a = n;
else
    [S,Q] = size(n);
    nanInd = any(isnan(n),1);
    a = zeros(S,Q,'like',n);
    [~,maxRows] = max(n,[],1);
    onesInd = maxRows + S*(0:(Q-1));
    a(onesInd) = 1;
    a(:,nanInd) = NaN;
end

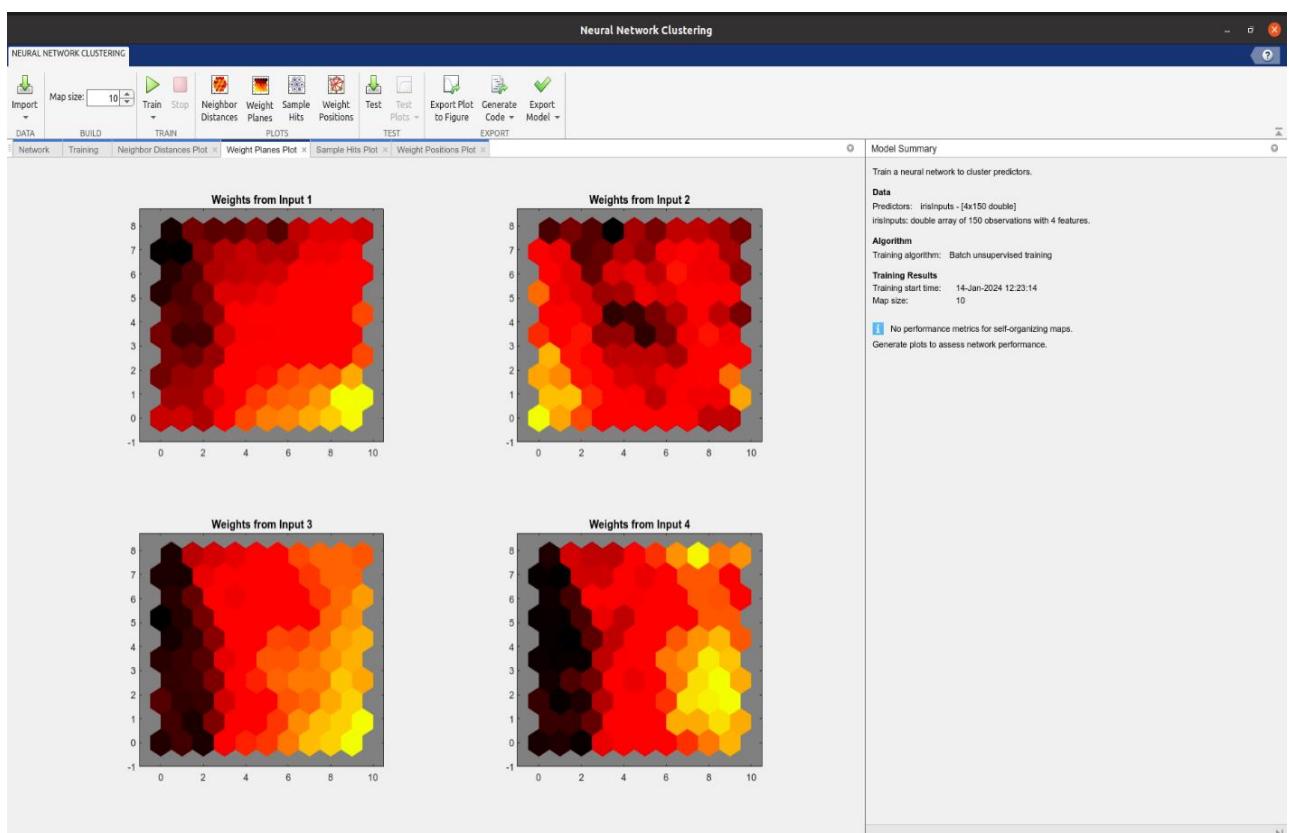
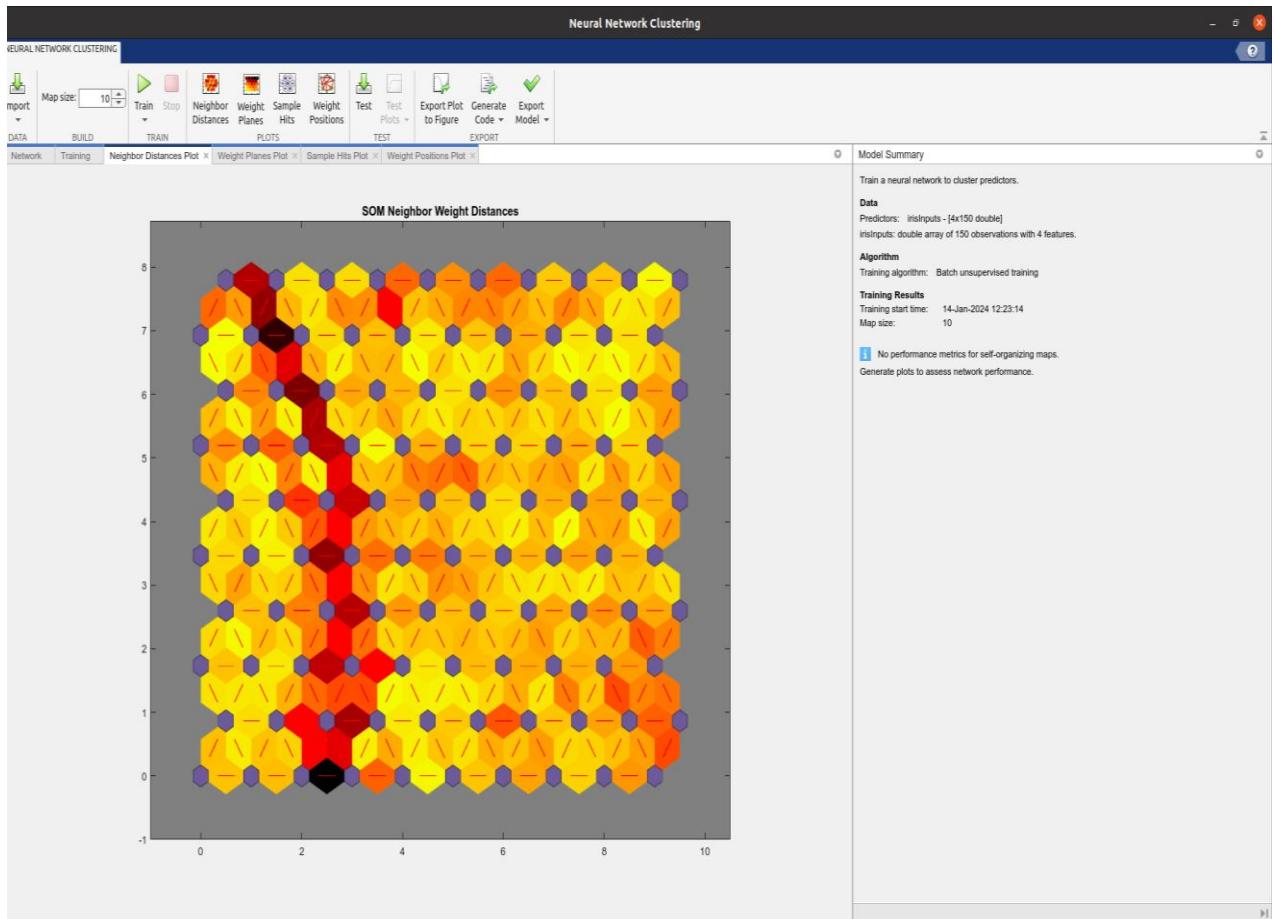
```

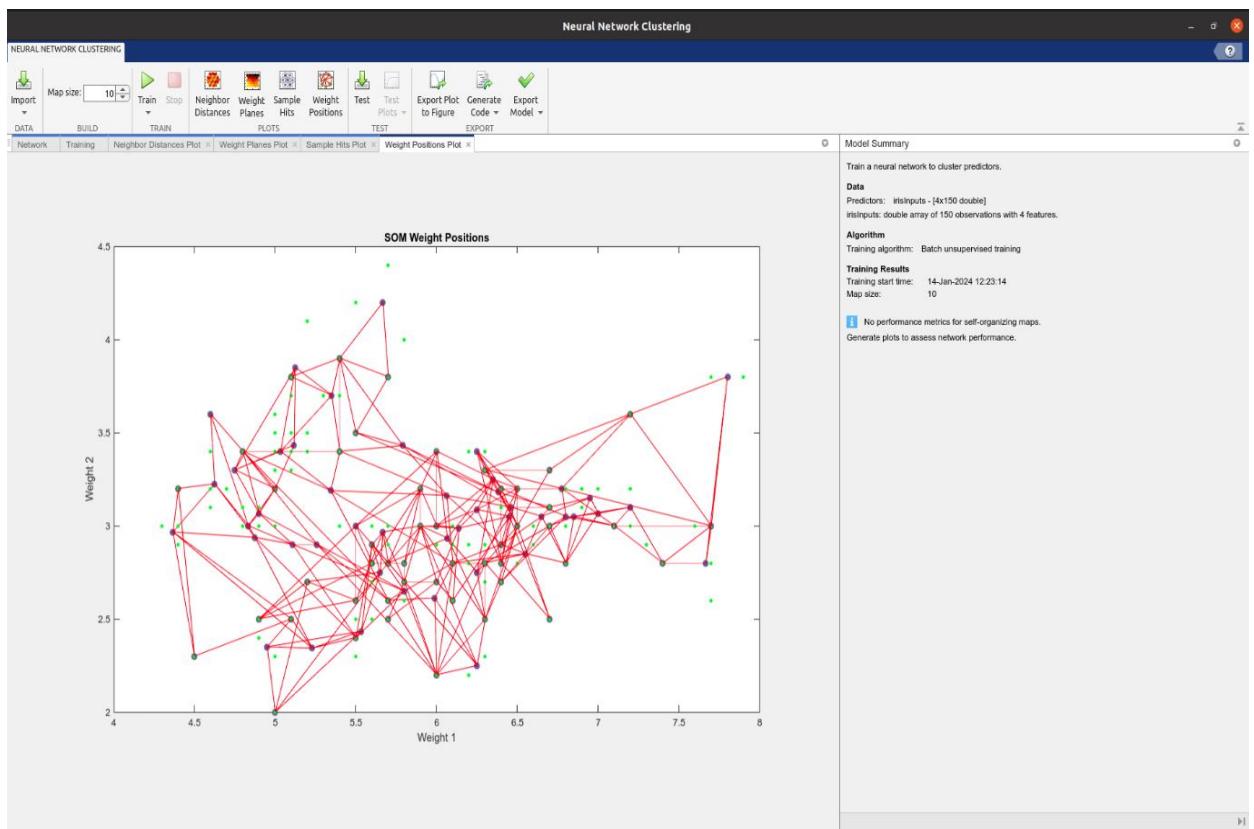
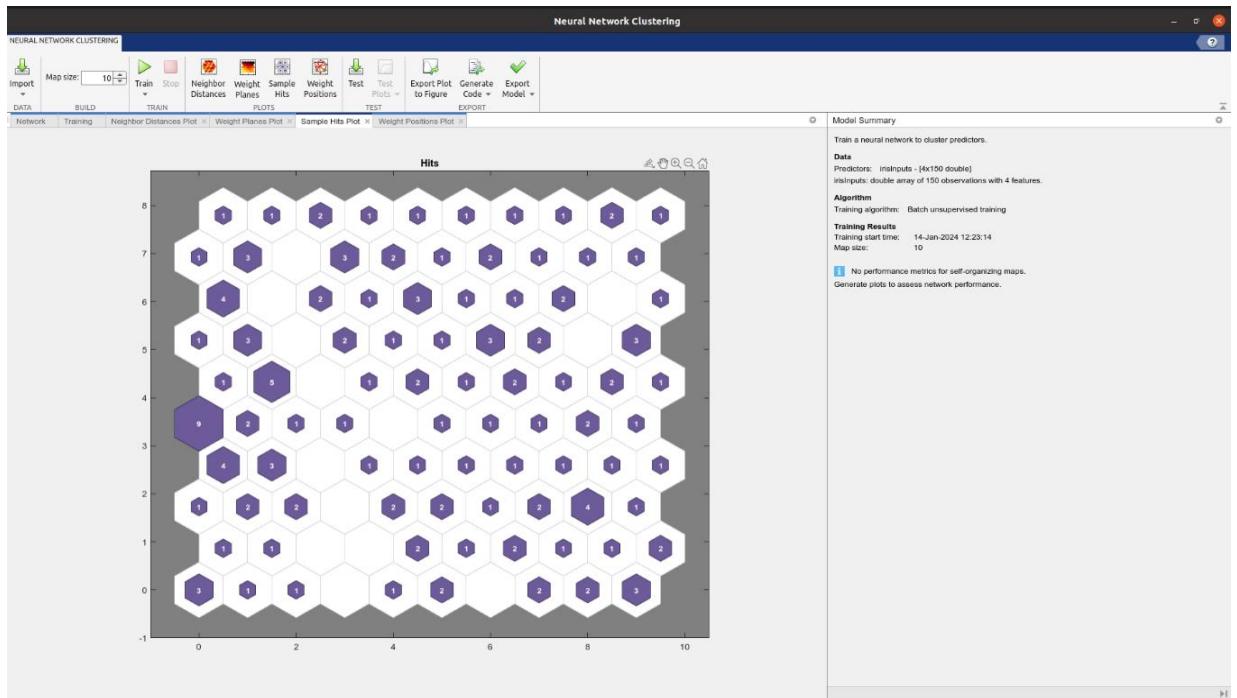
```
end  
end
```

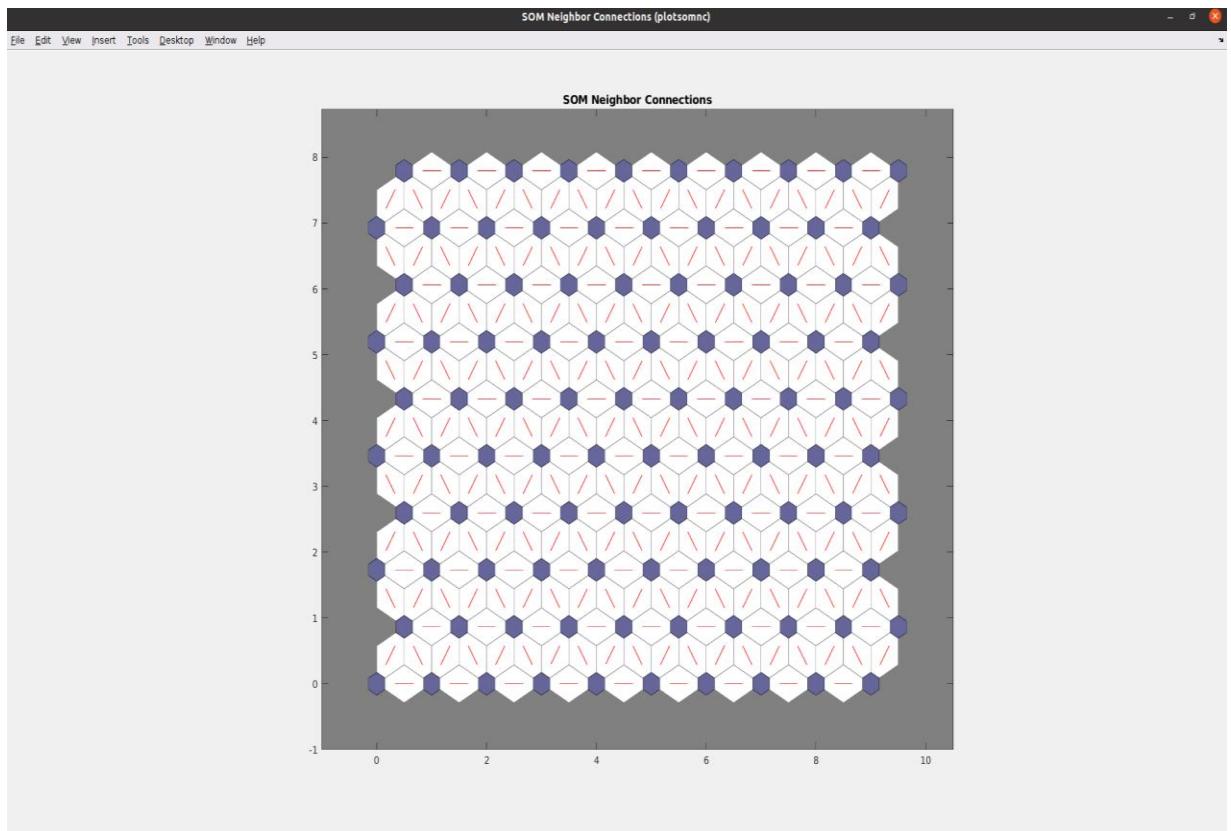












- Δημιουργία, εκπαίδευση και αξιολόγηση νευρωνικών δικτύων μέσω της εφαρμογής *ntstool* του Matlab(για Time Series)

Dataset: Simple NAR

Αρχείο: ntstool1.m

```
T = simpleNARTargets;

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging
% problems.
% 'trainscg' uses less memory. Suitable in low memory
% situations.
trainFcn = 'trainlm'; % Levenberg-Marquardt backpropagation.

% Create a Nonlinear Autoregressive Network
feedbackDelays = 1:2;
hiddenLayerSize = 10;
```

```

net = narnet(feedbackDelays,hiddenLayerSize,'open',trainFcn);

% Prepare the Data for Training and Simulation
% The function PREPARETS prepares timeseries data for a
particular network,
% shifting time by the minimum amount to fill input states and
layer
% states. Using PREPARETS allows you to keep your original
time series data
% unchanged, while easily customizing it for networks with
differing
% numbers of delays, with open loop or closed loop feedback
modes.
[x,xi,ai,t] = preparets(net,[],[],T);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t,xi,ai);

% Test the Network
y = net(x,xi,ai);
e = gsubtract(t,y);
performance = perform(net,t,y)

% View the Network
view(net)

% Plots
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotresponse(t,y)
%figure, ploterrcorr(e)
%figure, plotinerrcorr(x,e)

netc = closeloop(net);
netc.name = [net.name ' - Closed Loop'];
view(netc)
[xc,xic,aic,tc] = preparets(netc,[],[],T);
yc = netc(xc,xic,aic);
closedLoopPerformance = perform(net,tc,yc)

nets = removedelay(net);
nets.name = [net.name ' - Predict One Step Ahead'];
view(nets)
[xs,xis,ais,ts] = preparets(nets,[],[],T);

```

```

ys = nets(xs,xis,ais);
stepAheadPerformance = perform(nets,ts,ys)

```

Αρχείο: ntstool2.m

```

function [y1,xf1] = myNeuralNetworkFunction(x1,xil)

% ===== NEURAL NETWORK CONSTANTS =====

% Input 1
x1_step1.xoffset = 0.162182308193243;
x1_step1.gain = 2.38741093727801;
x1_step1.ymin = -1;

% Layer 1
b1 = [-0.78050535166348911087;-2.0849245192338132782;-
2.8785643178901474393;0.89766331609471228248;0.088976040943150
758178;-0.44088044139418058887;-0.73581130670817540551;-
0.95103927214832273407;-3.4232842496954440925;-
5.6182386988462162947];
IW1_1 = [0.25649372671907622045
0.51306587587460850131;0.99039191925942049455 -
4.4165174406823366482;2.9822327072348175925 -
2.9984809989611913572;-2.5506368246075159512
3.7600706728469095452;-4.8183612741607850793
0.80615057471338102957;-2.4543392703677011646 -
0.66941920036414126471;-4.1759717946387926446
1.4773697178110323236;-0.31233797708968025209 -
0.62586427485519002811;-3.9501011793521545457
2.0869667894484149606;-2.7671126974044439883 -
2.5346962515515634173];

% Layer 2
b2 = -2.5872014156057723255;
LW2_1 = [-4.1909607108564701505 -0.00017113419932886507783
0.00044023366310221820645 0.00013488906020620426859 -
0.00026635585721289872048 -0.000320214617603362475
0.00017808334703919804706 -3.1438488092274465835
8.1416521093992210009e-05 1.51164601487840633];

% Output 1
y1_step1.ymin = -1;
y1_step1.gain = 2.38741093727801;
y1_step1.xoffset = 0.162182308193243;

% ===== SIMULATION =====

```

```

% Dimensions
TS = size(x1,2); % timesteps

% Input 1 Delay States
xd1 = mapminmax_apply(xil,x1_step1);
xd1 = [xd1 zeros(1,1)];

% Allocate Outputs
y1 = zeros(1,TS);

% Time loop
for ts=1:TS

    % Rotating delay state position
    xdts = mod(ts+1,3)+1;

    % Input 1
    xd1(:,xdts) = mapminmax_apply(x1(:,ts),x1_step1);

    % Layer 1
    tapdelay1 = reshape(xd1(:,mod(xdts-[1 2]-1,3)+1),2,1);
    a1 = tansig_apply(b1 + IW1_1*tapdelay1);

    % Layer 2
    a2 = b2 + LW2_1*a1;

    % Output 1
    y1(:,ts) = mapminmax_reverse(a2,y1_step1);
end

% Final delay states
finalxts = TS+(1: 2);
xits = finalxts(finalxts<=2);
xts = finalxts(finalxts>2)-2;
xf1 = [xil(:,xits) x1(:,xts)];
end

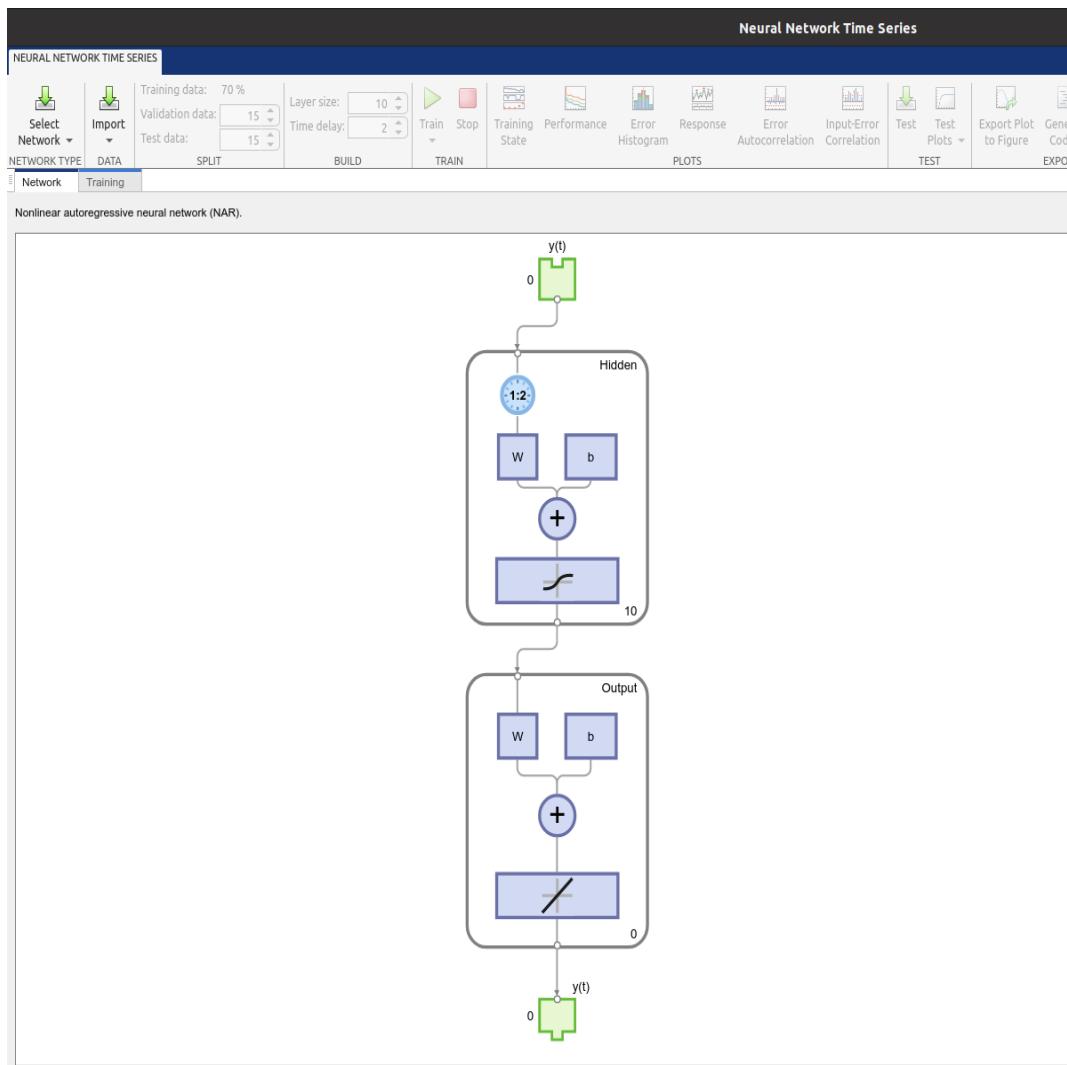
% ===== MODULE FUNCTIONS =====

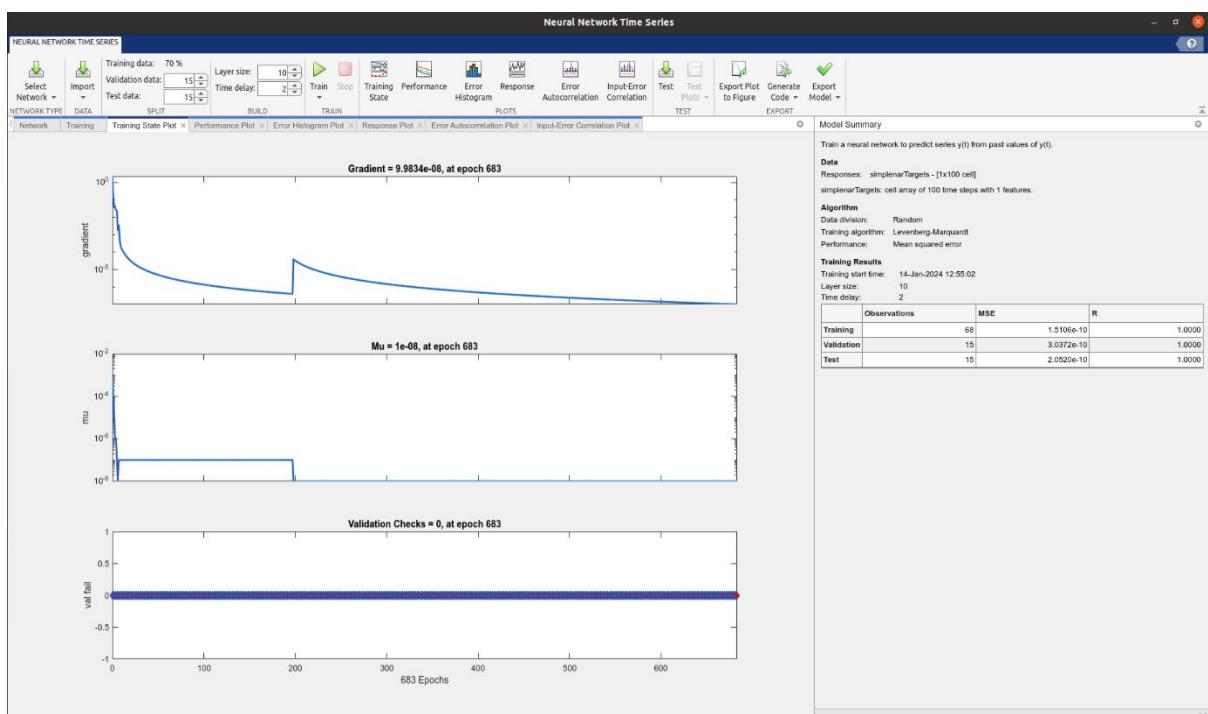
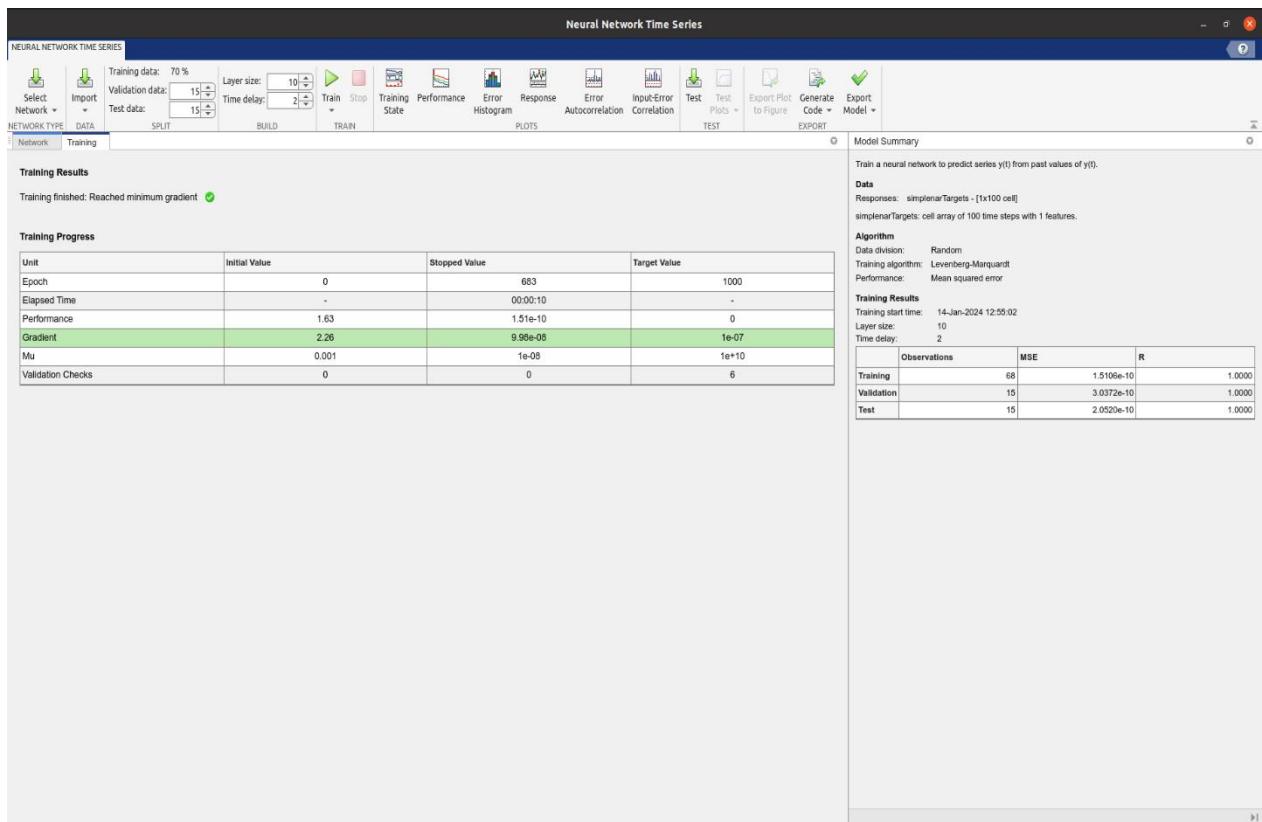
% Map Minimum and Maximum Input Processing Function
function y = mapminmax_apply(x,settings)
y = bsxfun(@minus,x,settings.xoffset);
y = bsxfun(@times,y,settings.gain);
y = bsxfun(@plus,y,settings.ymin);
end

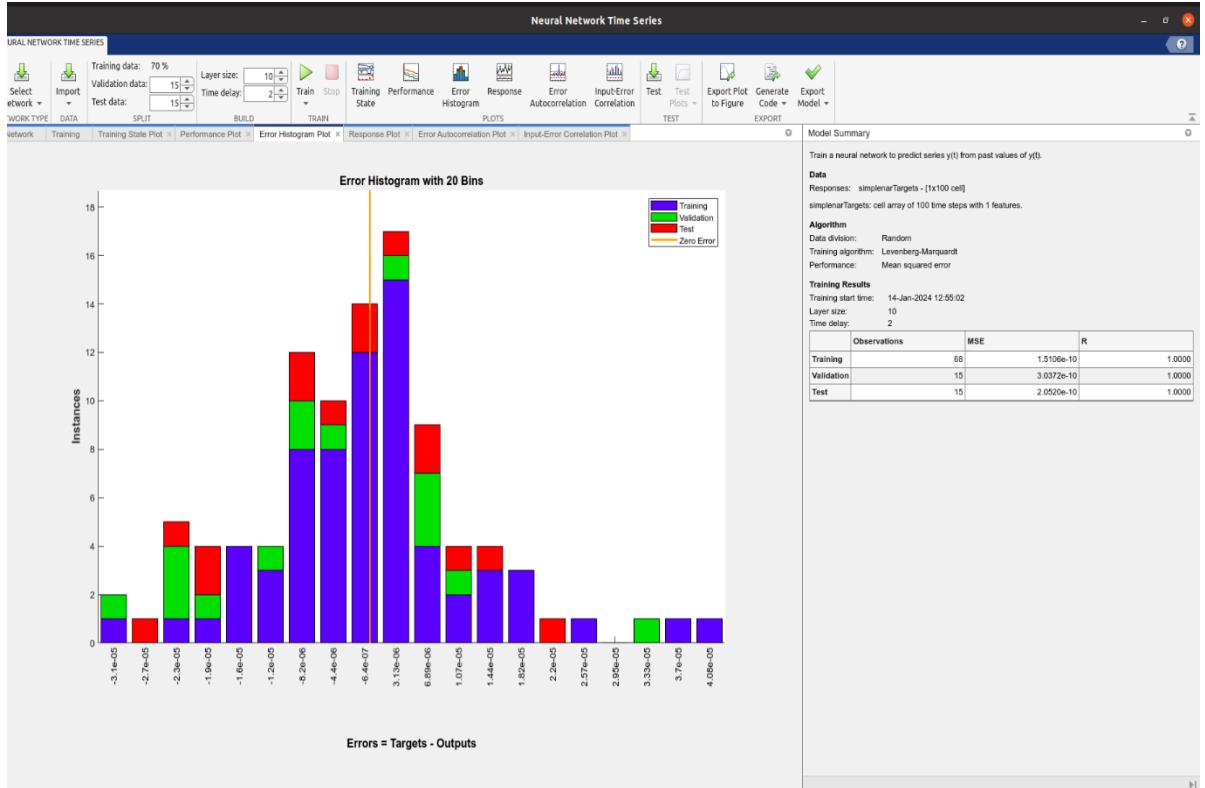
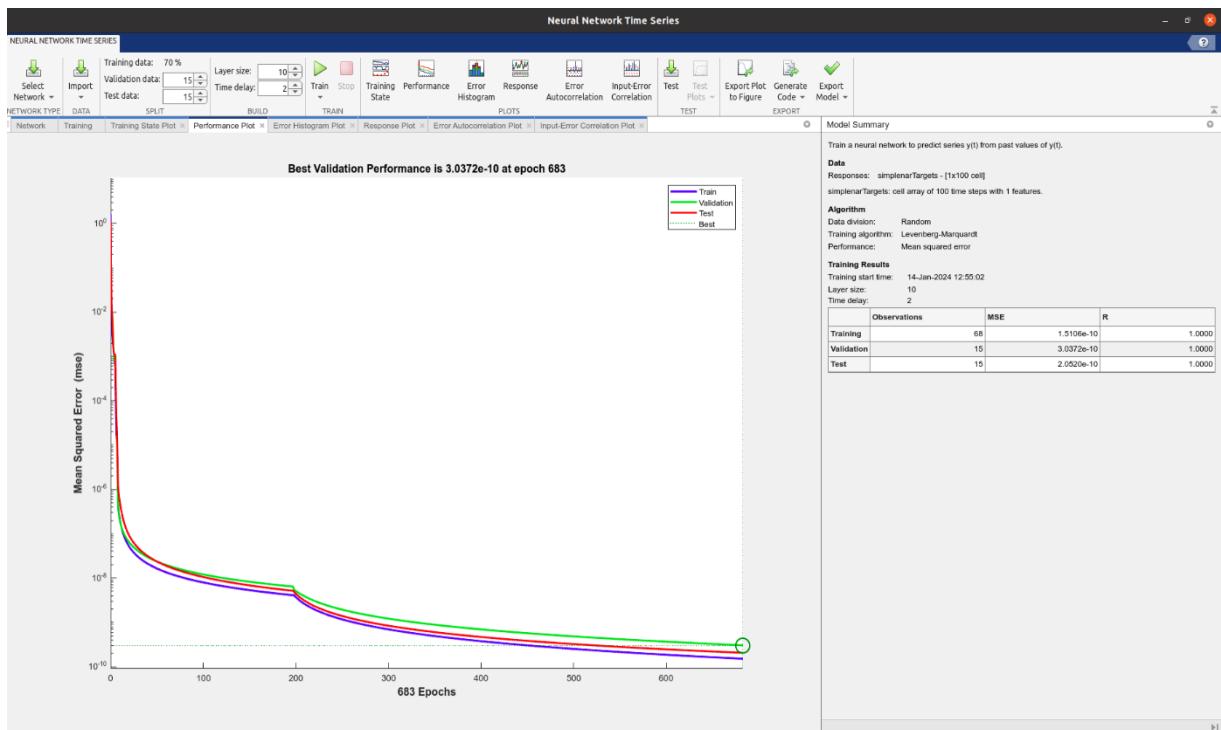
% Sigmoid Symmetric Transfer Function
function a = tansig_apply(n,~)
a = 2 ./ (1 + exp(-2*n)) - 1;
end

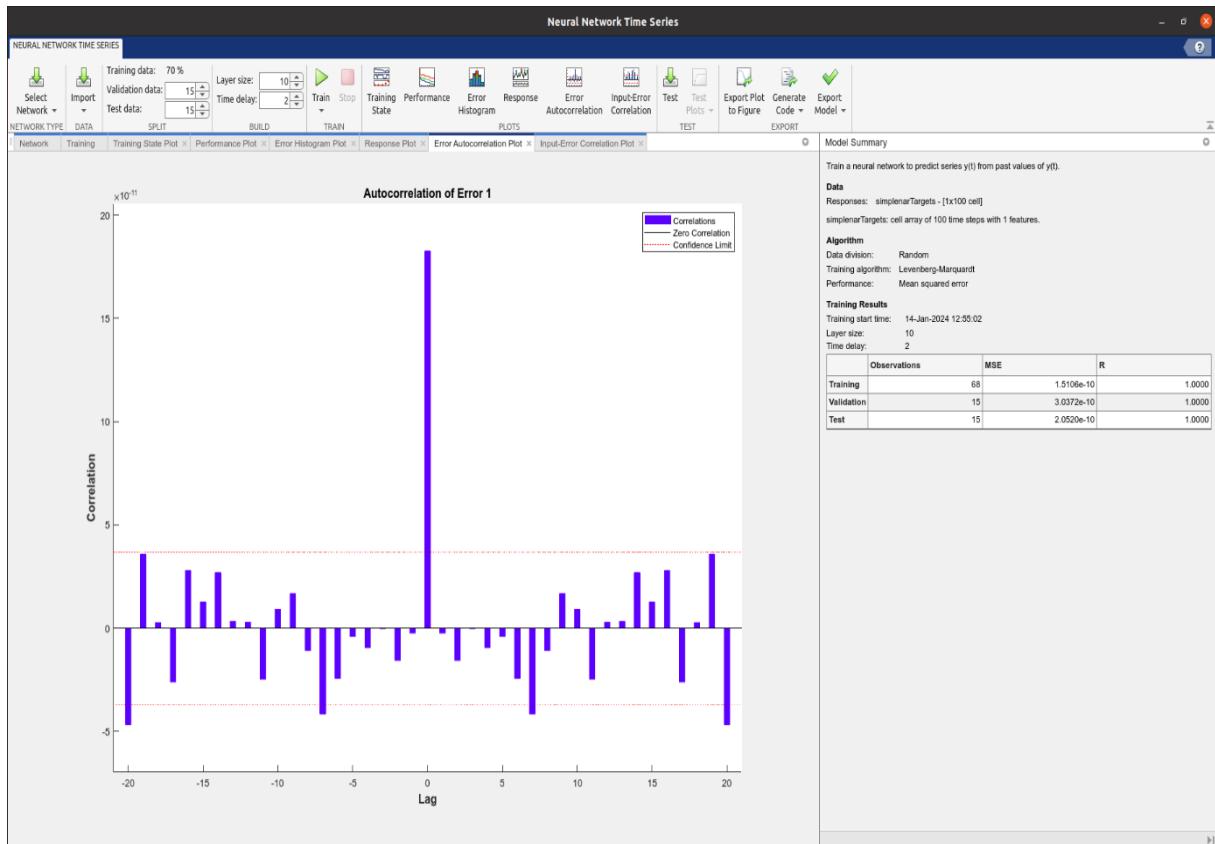
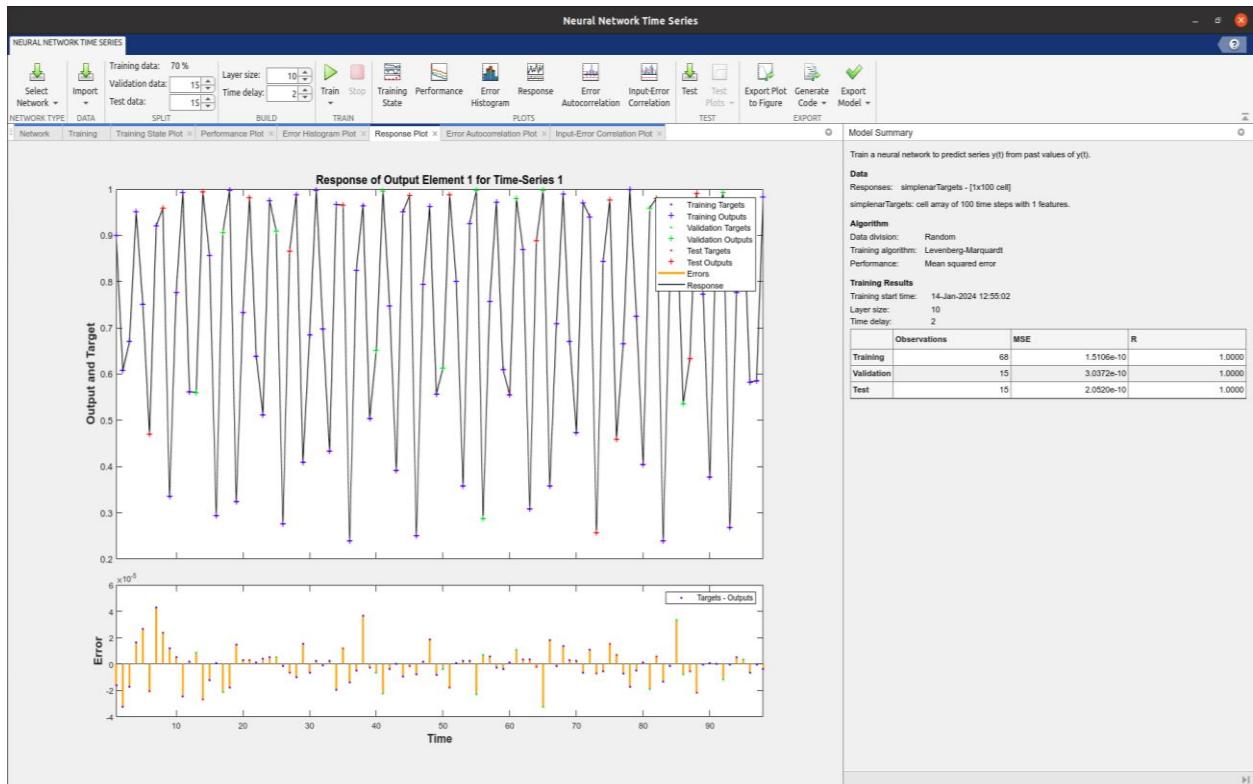
```

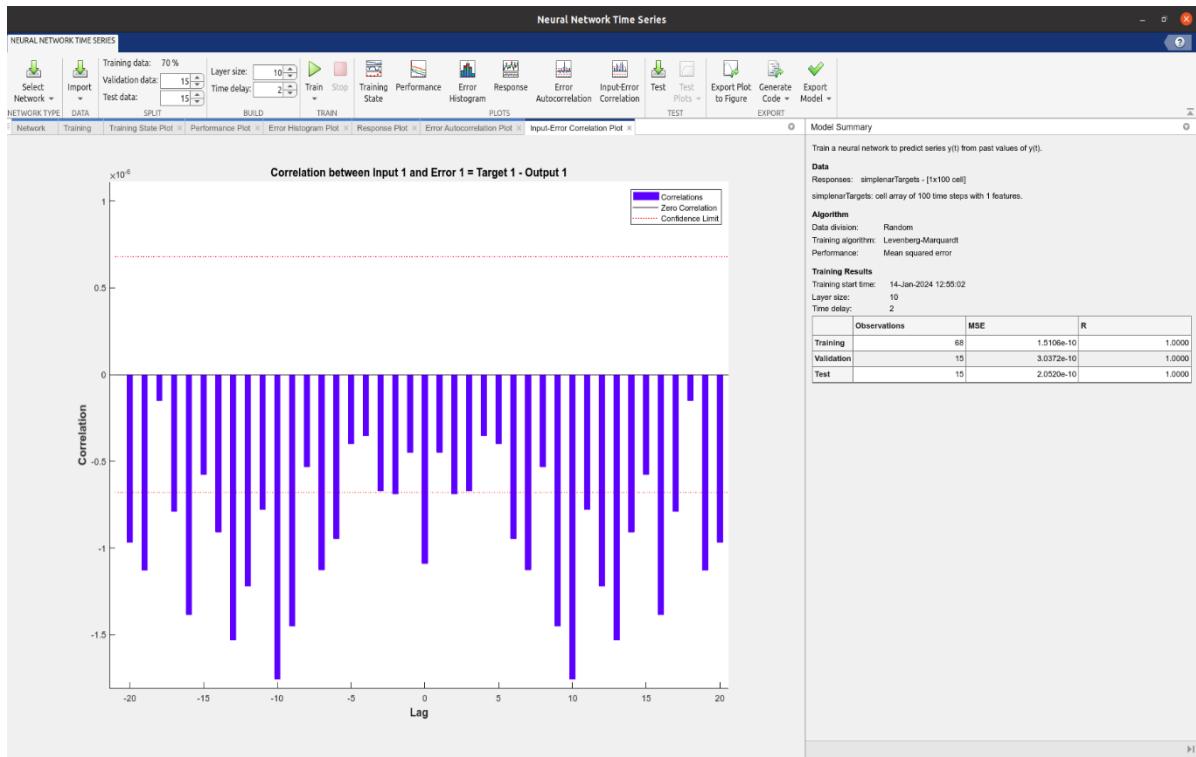
```
% Map Minimum and Maximum Output Reverse-Processing Function
function x = mapminmax_reverse(y,settings)
x = bsxfun(@minus,y,settings.ymin);
x = bsxfun(@rdivide,x,settings.gain);
x = bsxfun(@plus,x,settings.xoffset);
```











Σχολιασμός αποτελεσμάτων:

Στο ερώτημα Α της εργασίας δημιουργούμε νευρωνικά δίκτυα με τη βοήθεια 2 εφαρμογών GUI του Matlab. Αναλυτικότερα, χρησιμοποιούμε τα εξής: nctool για Clustering και ntstool για Time-series analysis. Αντίστοιχα, χρησιμοποιούμε τα εξής datasets που υπάρχουν έτοιμα στο Matlab: Iris flowers, Simple Nar. Μέσω των παραπάνω διεπαφών εκπαιδεύουμε τα δίκτυα που δημιουργούμε και οδηγούμαστε σε ενδιαφέροντα συμπεράσματα, μέσω των διαγραμμάτων που δημιουργούνται.

Ειδικότερα, στο 1^o παράδειγμα παρουσιάζονται τα εξής διαγράμματα: SOM Neighbor Weight Distances, weights from every input, hits, SOM weight positions και SOM neighbor connections. Συνοπτικά, παρουσιάζονται δηλαδή οι θέσεις των βαρών των νευρώνων στο χάρτη SOM, το πόσο απέχουν τα βάρη στο χάρτη SOM, τα βάρη που έχουν ανατεθεί σε κάθε είσοδο και τις συνδέσεις γειτονίας μεταξύ νευρώνων στο χάρτη SOM.

Στο 2^o παράδειγμα έχουμε τα εξής γραφήματα: Gradient, Mu, Validation Checks, best validation performance, error histogram. Επίσης, βλέπουμε τη σχέση Error-Time, το Response of output element 1 for Time Series 1, το Autocorrelation of Error 1 και το Target 1 – Output 1. Απεικονίζεται δηλαδή η διαφορά μεταξύ της επιθυμητής και της πραγματικής εξόδου για την 1^η χρονοσειρά, η αλλαγή της εξόδου 1 σε σχέση με την 1^η χρονοσειρά και η συσχέτιση του σφάλματος για την 1^η χρονοσειρά με τον εαυτό του.

Μέρος Β

Deep Network Designer

Σημείωση: Τα B1 και B2 παρουσιάζονται ως 1. Απλά, γίνεται παρουσίαση 2 διαφορετικών παραδειγμάτων.

Παράδειγμα 1

Βήματα:

1. Άνοιγμα του App Network Designer
2. New Blank Network
3. Διαλέγουμε από το πλαίσιο αριστερά της οθόνης τα κατάλληλα layers, και όπου κρίνεται απαραίτητο αλλάζουμε τις παραμέτρους του κάθε στρώματος μέσω του πλαισίου δεξιά της οθόνης.

Στρώματα:

imageInput (το παράδειγμα ασχολείται με εικόνες)

conv

relu

conv

relu

fc

fc

softmax (μέτρηση πιθανοτήτων)

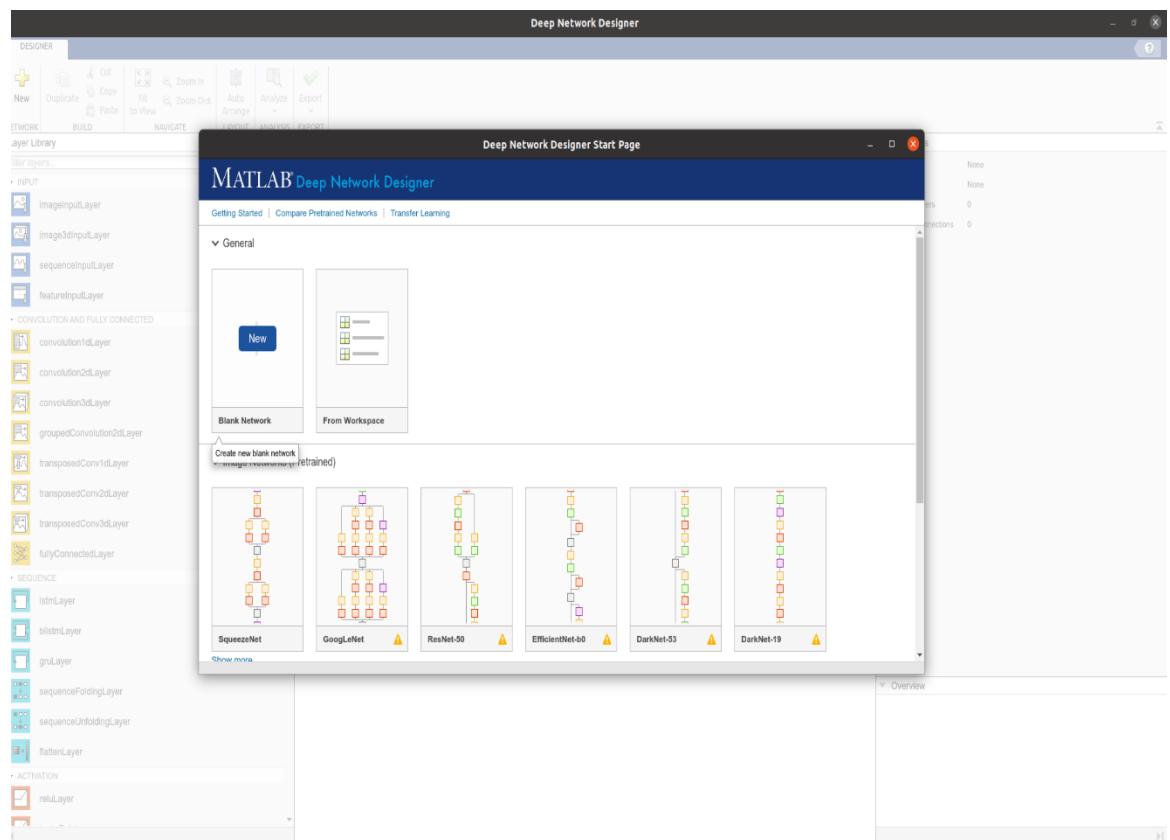
classoutput (πρόβλημα κατηγοριοποίησης)

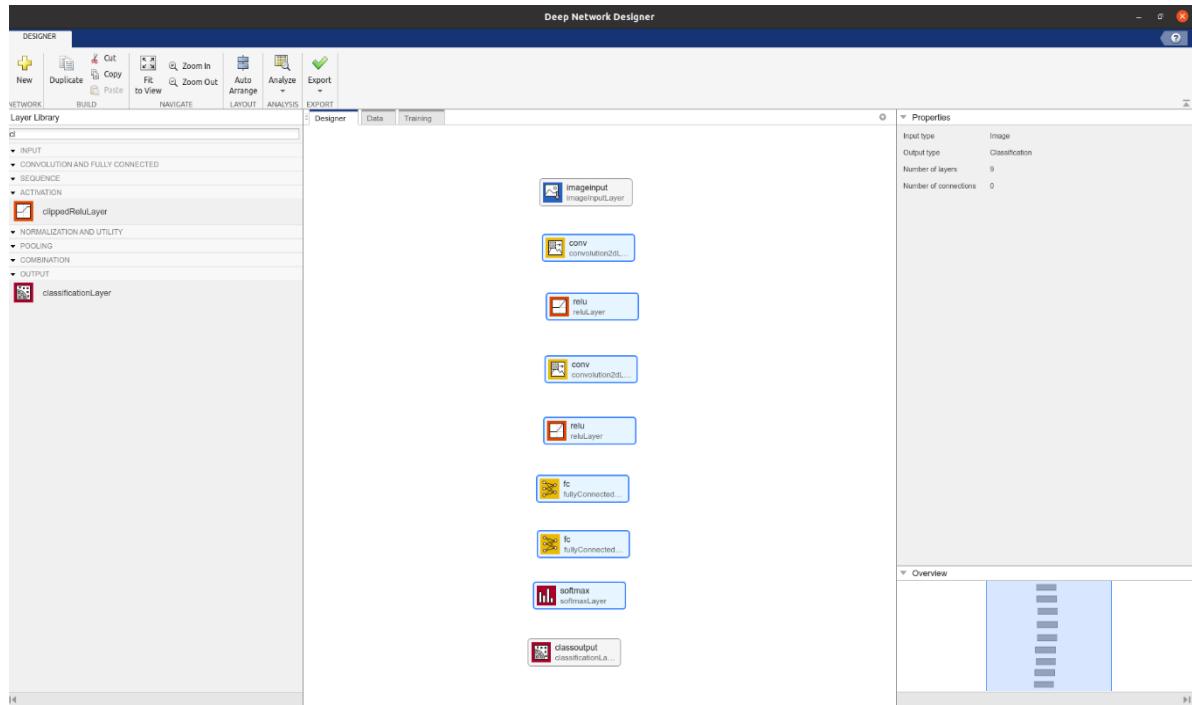
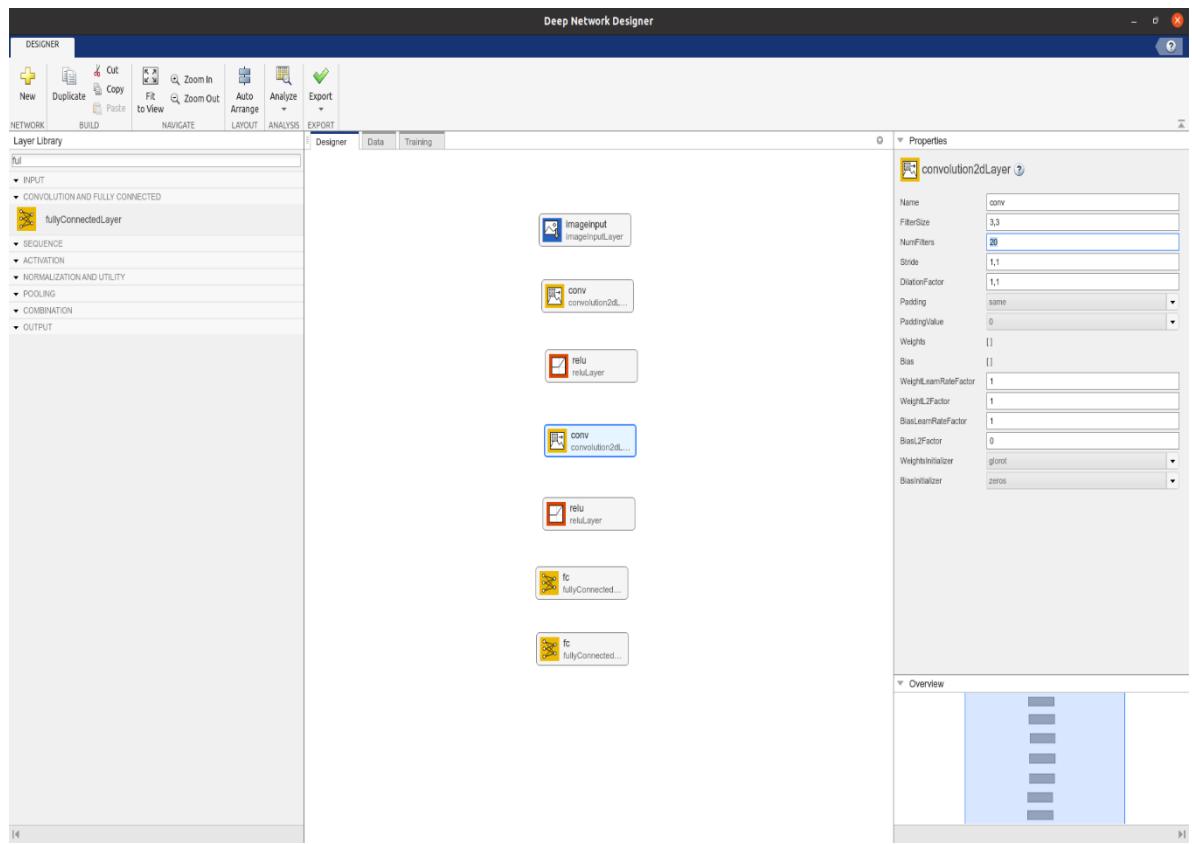
Μεταξύ του στρώματος εισόδου και του στρώματος εξόδου, θα μπορούσε να υπάρχει οποιοσδήποτε συνδυασμός στρωμάτων. Το στρώμα εξόδου: 10 έξοδοι, καθώς ασχολούμαστε με πρόβλημα αναγνώρισης μονοψήφιων αριθμών(0-9).

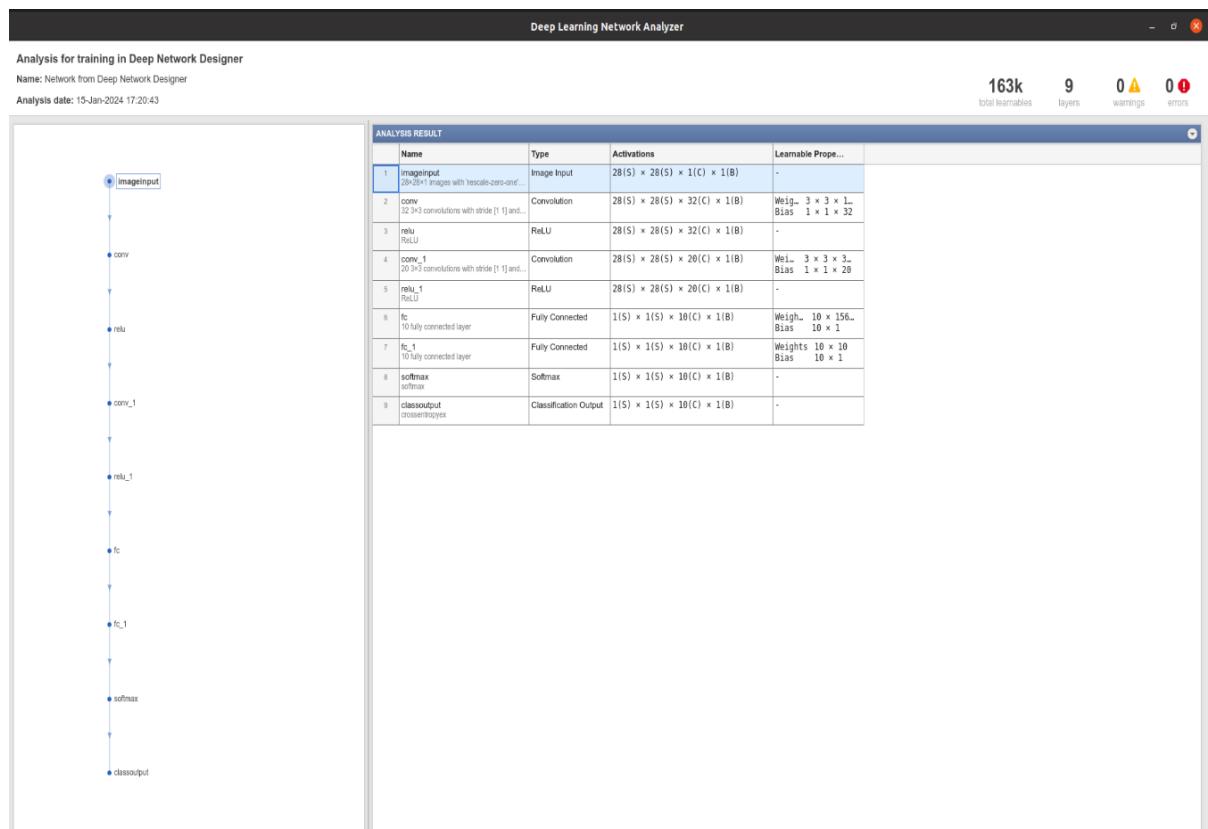
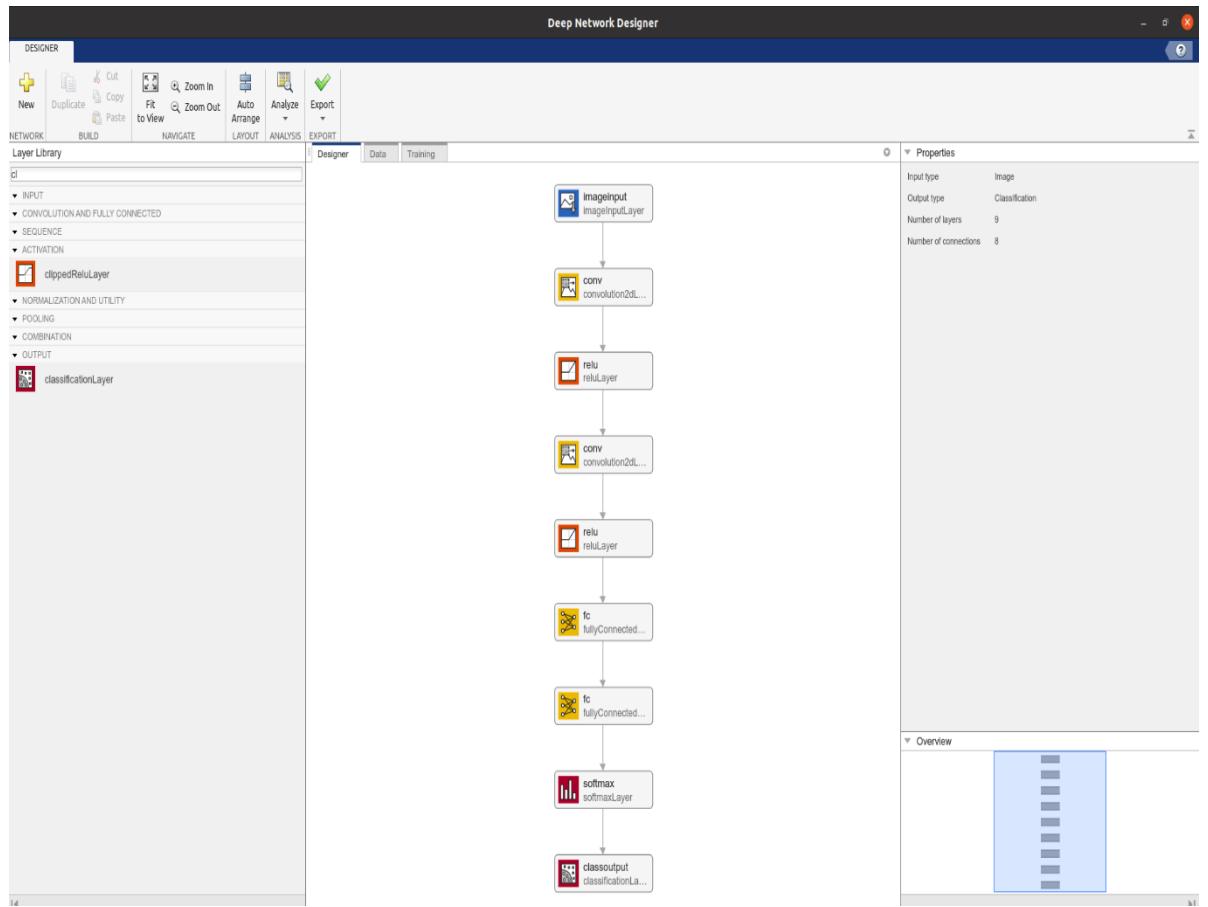
4. Σύνδεση των στρωμάτων μεταξύ τους
5. Κλικ στο Auto Arrange
6. Κλικ στο Analyze → Παρατηρούμε ότι ως αποτέλεσμα έχουμε 0 errors και 0 warnings. Άρα, η διαδικασία συνεχίζεται κανονικά.
7. Export
 - a. Network
 - b. Generate code
8. Επιλογή του Data tab → Import data:
Browse→Home/Downloads/Matlab/R2022a/toolbox/nnet/nndemos/nndatasets/DigitDataset → Import
9. Επιλογή του Training tab → Train
10. Export

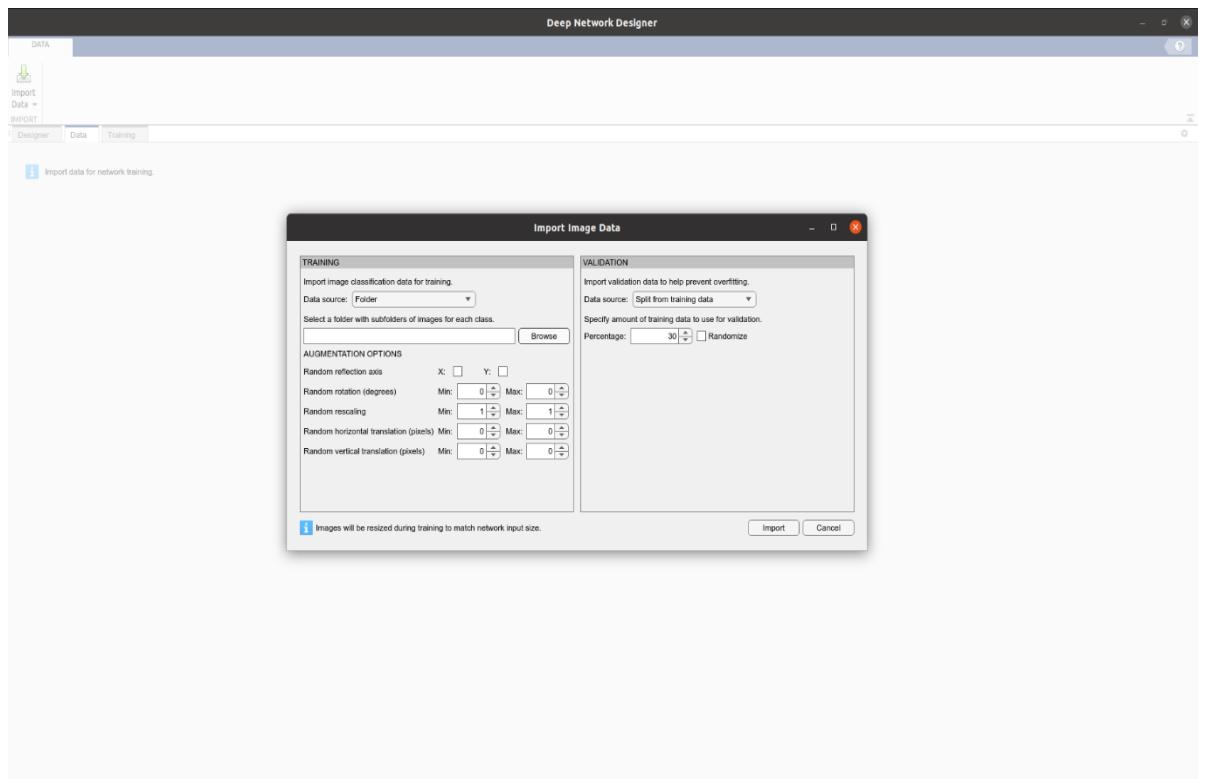
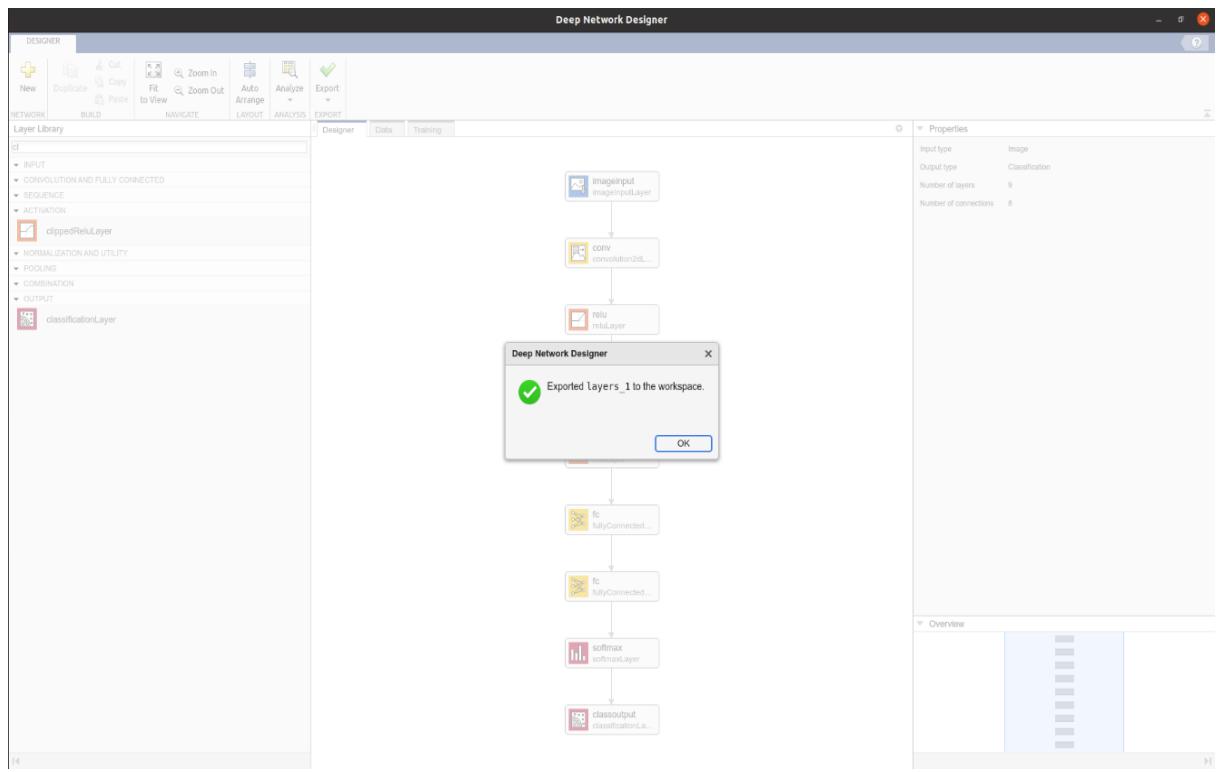
- Δημιουργία του trainedNetwork_1
- Δημιουργία του trainInfoStruct_1, που παρέχει στο χρήστη χρήσιμα στατιστικά που αφορούν τη διαδικασία της εκπαίδευσης.

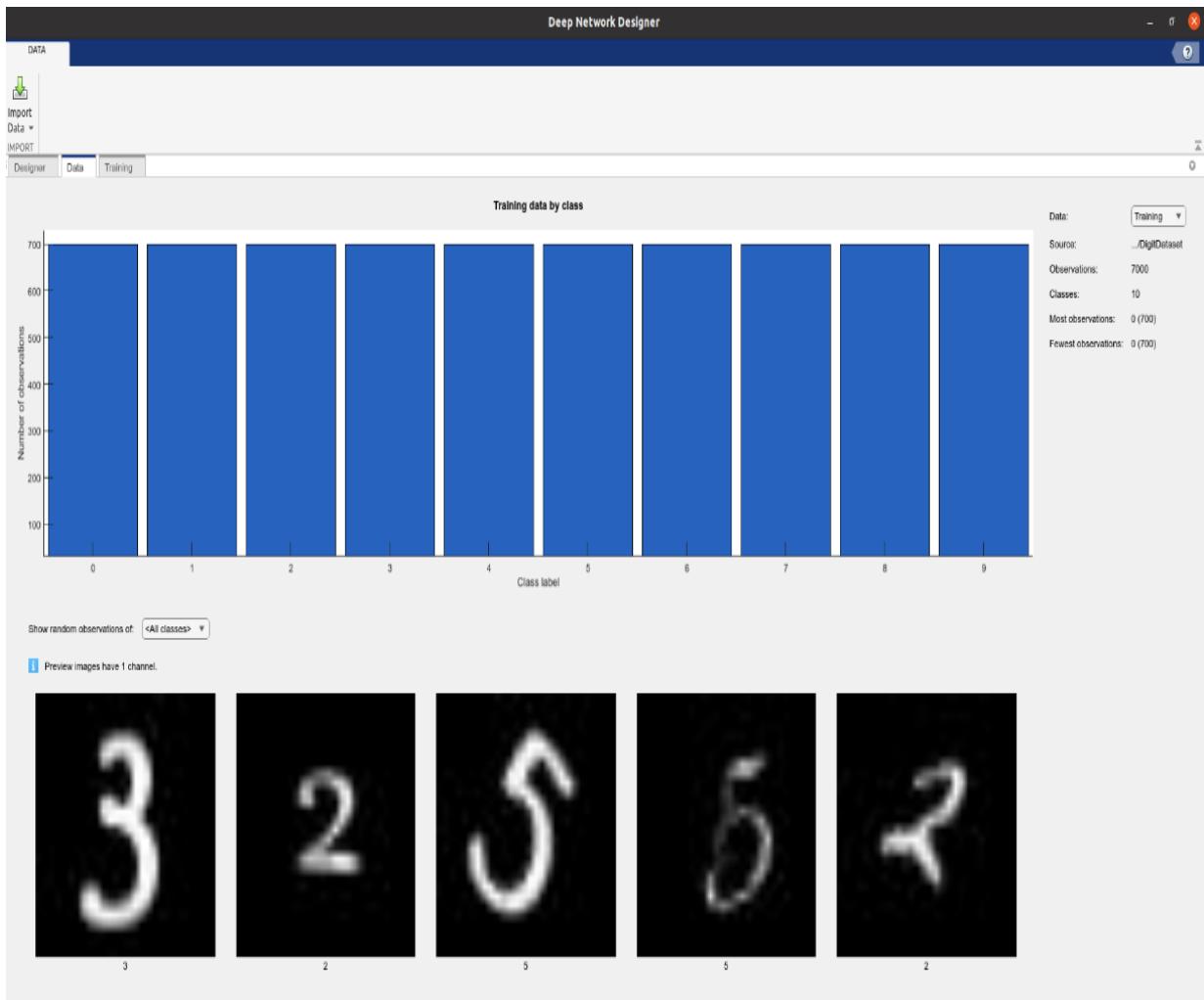
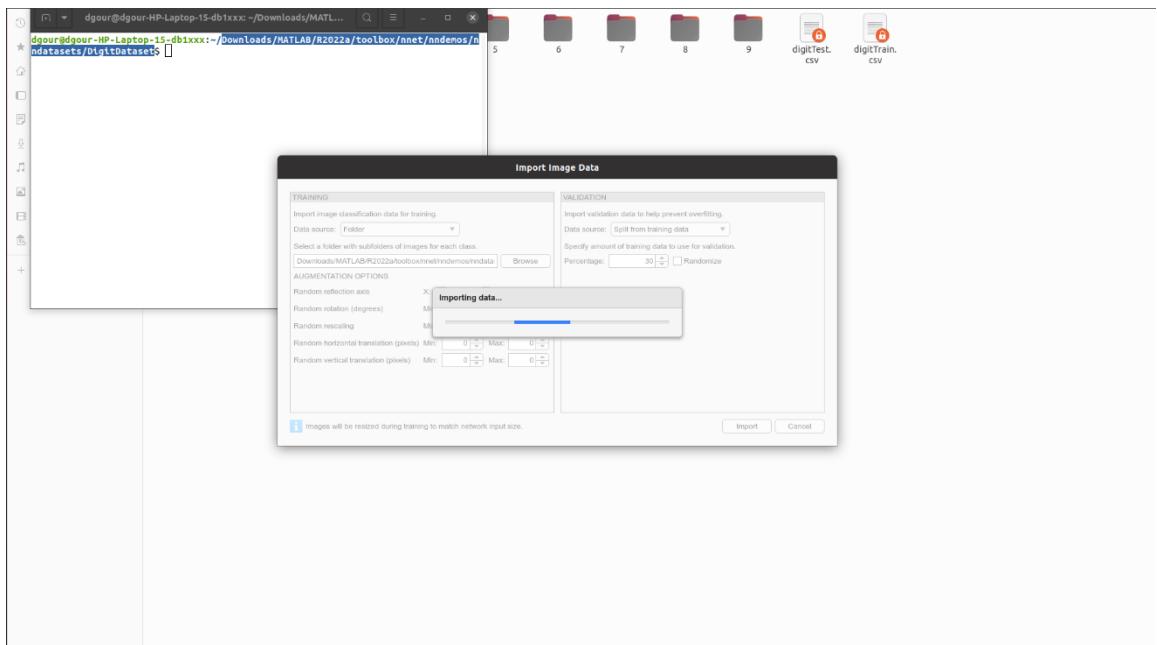
Ενώ, στο trainedNetwork_1 μπορούμε να δούμε αναλυτικά πληροφορίες για το εκάστοτε στρώμα και τις τιμές των Weights, Bias και άλλων χρήσιμων παραμέτρων.













Σχολιασμός αποτελεσμάτων:

Στο συγκεκριμένο παράδειγμα δημιουργούμε «από το 0» ένα νευρωνικό δίκτυο, επιλέγοντας κάθε στρώμα του, με στόχο να αναγνωρίσουμε χαρακτηριστικά από εικόνες. Όπως τονίσαμε και παραπάνω έγινε χρήση της εφαρμογής Deep Network Designer και θα γίνει εκπαίδευση, αλλά και έλεγχος της απόδοσης του δικτύου πάνω στο DigitDataset, που είναι προ εγκαταστημένο κατά την εγκατάσταση του Matlab, και περιέχει φωτογραφίες μονοψήφιων αριθμών(0-9). Σε κάθε ψηφίο αντιστοιχούν 700 εικόνες. Όσον αφορά τα στρώματα του δικτύου:

- **imageInput:** Το πρώτο στρώμα (imageInput) είναι υπεύθυνο για την εισαγωγή των εικόνων στο δίκτυο. Αυτό το στρώμα καθορίζει το μέγεθος των εικόνων εισόδου.
- **conv:** Το στρώμα συνέλιξης (conv) εφαρμόζει συνελίξεις στις εικόνες. Αυτό βοηθά στην εξαγωγή χαρακτηριστικών από τις εικόνες.
- **relu:** Το στρώμα ενεργοποίησης relu εφαρμόζει τη συνάρτηση ενεργοποίησης, η οποία προσθέτει μη-γραμμικότητα στο μοντέλο.
- **fc:** Το στρώμα πλήρως συνδεδεμένων νευρώνων (fc) εκτελεί πράξεις πλήρως συνδεδεμένου επιπέδου, συνδέοντας όλες τις εικόνες
- του προηγούμενου στρώματος με κάθε νευρώνα.
- **softmax:** Το στρώμα Softmax εφαρμόζει την συνάρτηση Softmax, μετατρέποντας τις εξόδους του προηγούμενου επιπέδου σε πιθανότητες.
- **classoutput:** Το στρώμα classoutput χρησιμοποιείται σε προβλήματα κατηγοριοποίησης. Εδώ γίνεται η τελική αναγνώριση της κλάσης.

Η συγκεκριμένη σειρά τοποθέτησης των στρωμάτων του νευρωνικού δικτύου χρησιμοποιείται σε προβλήματα εικόνας και κατηγοριοποίησης.

Σαν screenshot παρουσιάζεται το σύνολο των δεδομένων που αναμένεται να εκπαιδευτεί. Έπειτα, παρουσιάζεται η εκπαίδευση του νευρωνικού δικτύου σε κάθε εποχή. Αρχικά, η απόδοση του δικτύου είναι χαμηλή, ενώ τα σφάλματα πολλά, στη συνέχεια η κατάσταση διαφοροποιείται. Δηλαδή αυξάνεται αρκετά η απόδοση του δικτύου και μειώνεται συνεχώς ο αριθμός των λανθασμένων κατηγοριοποιήσεων. Είναι κατανοητό το γεγονός ότι αν επιλέγαμε περισσότερες εποχές εκπαίδευσης θα οδηγούμασταν σε καλύτερη απόδοση του δικτύου.

Παράδειγμα 2

Βήματα:

1. `openExample('nnet/TransferLearningWithDeepNetworkDesignerExample')`
2. `MerchData = unzip("MerchData.zip")`
3. Ανοιγμα του App DeepNetworkDesigner
4. Ανοιγμα του προεκπαιδευμένου νευρωνικού δικτύου SqueezeNet
5. Import των MerchData, και αλλαγή των παραμέτρων
6. Ανοιγμα του Data tab
7. Προετοιμασία του δικτύου για εκπαίδευση:

Παρατηρούμε ότι τα MerchData αναγνωρίζουν 5 κλάσεις. Οπότε, πρέπει να επεξεργαστούμε το τελευταίο classification layer. Συγκεκριμένα, για το SqueezeNet το τελευταίο classification layer είναι το τελευταίο conv, όπου θα αλλάξουμε τις παραμέτρους του(NumFilters=5). Η συγκεκριμένη παράμετρος-ιδιότητα καθορίζει τον αριθμό των κλάσεων για προβλήματα ταξινόμησης.

Επίσης, αλλάζουμε και το τελευταίο layer του δίκτυου, δηλαδή το classificatonLayer.

8. Analyze Network(για τσεκάρισμα λαθών στο δίκτυο)
9. Άνοιγμα του Training tab και αλλαγή ορισμένων Training Options.
10. Export
11. Classify new image

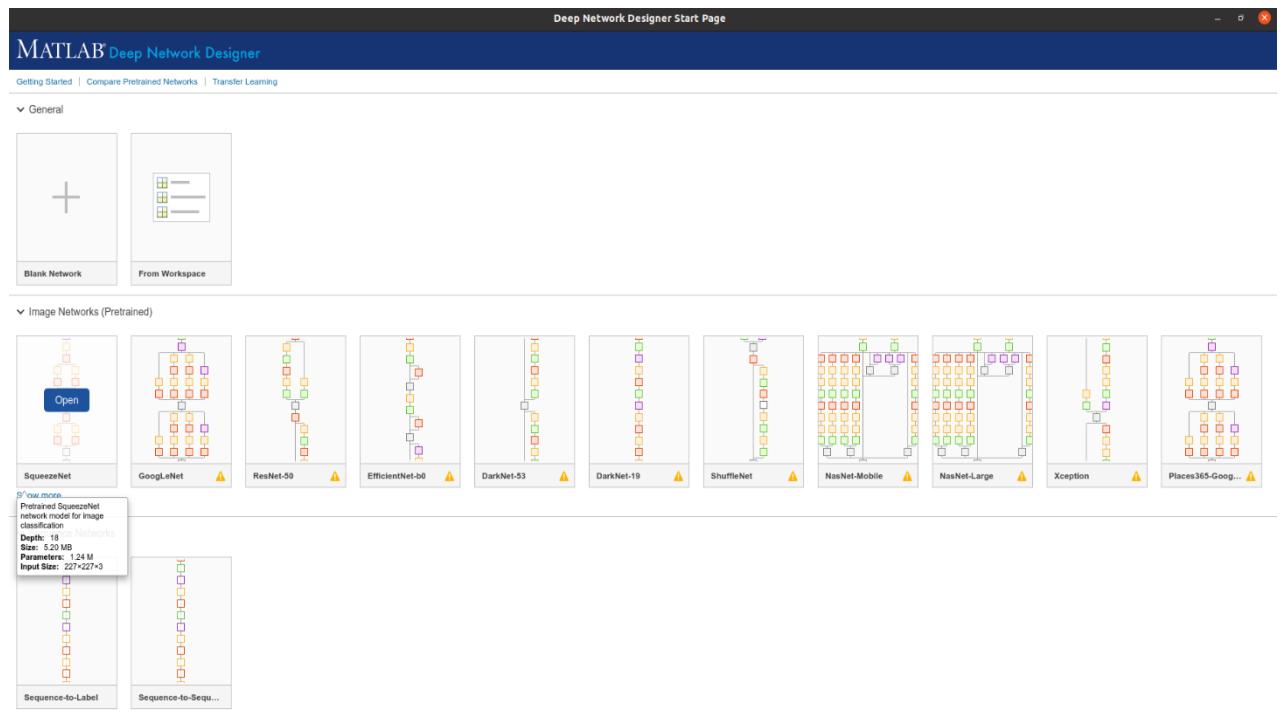
Αρχείο: b2_code.m

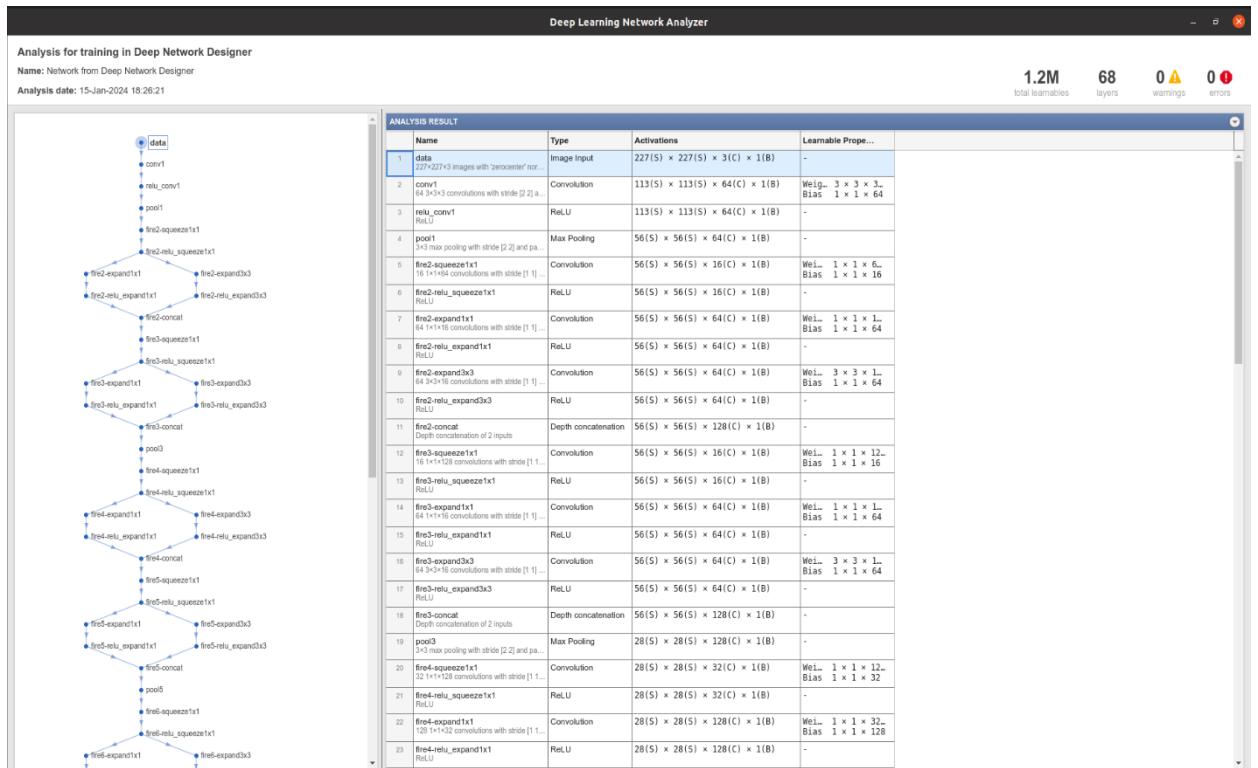
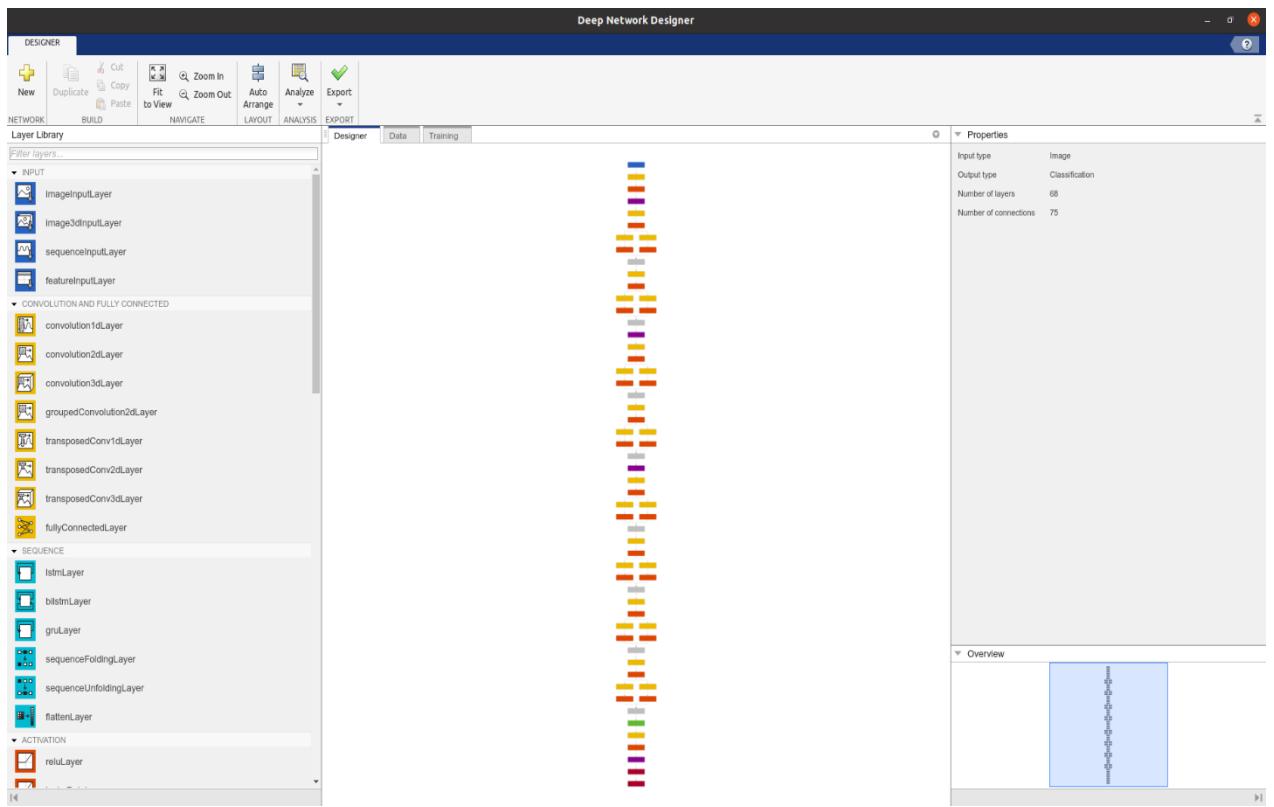
```
MerchData = unzip("MerchData.zip");

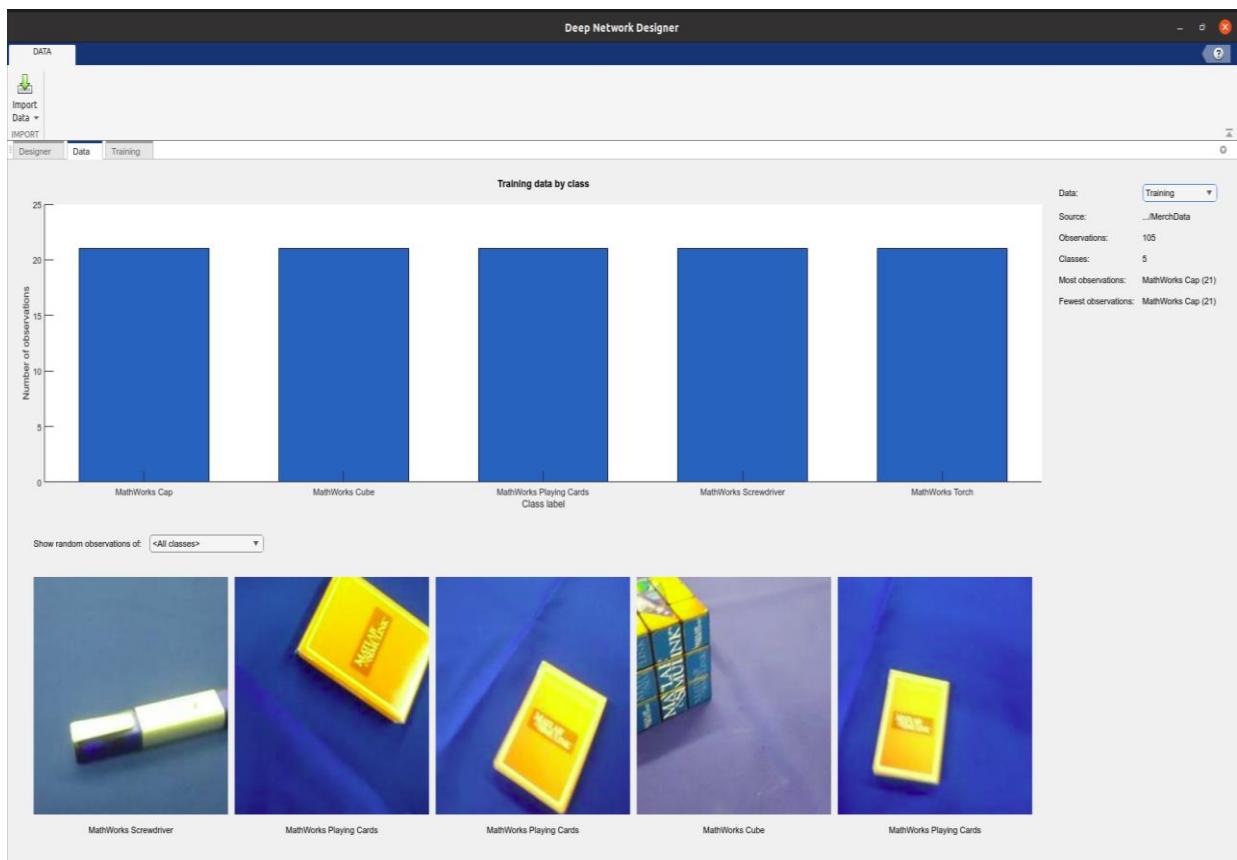
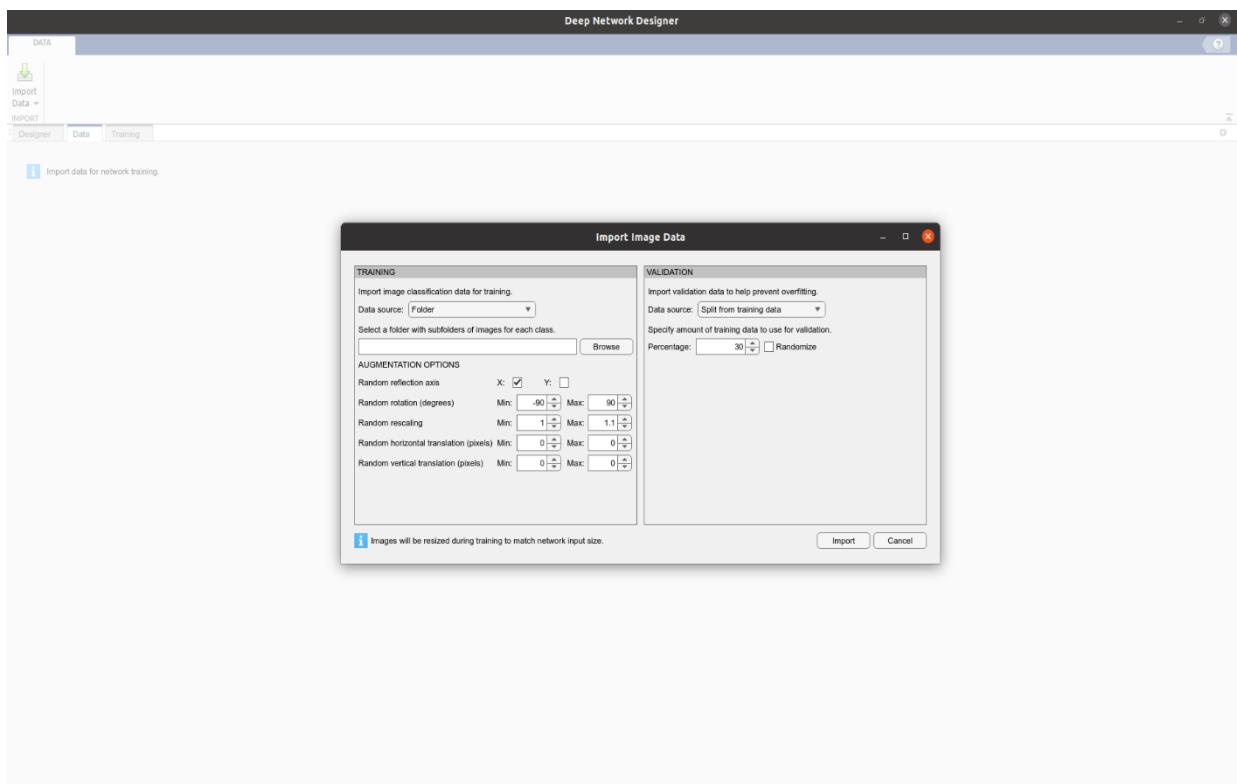
I = imread("MerchDataTest.jpg");
I = imresize(I, [227 227]);

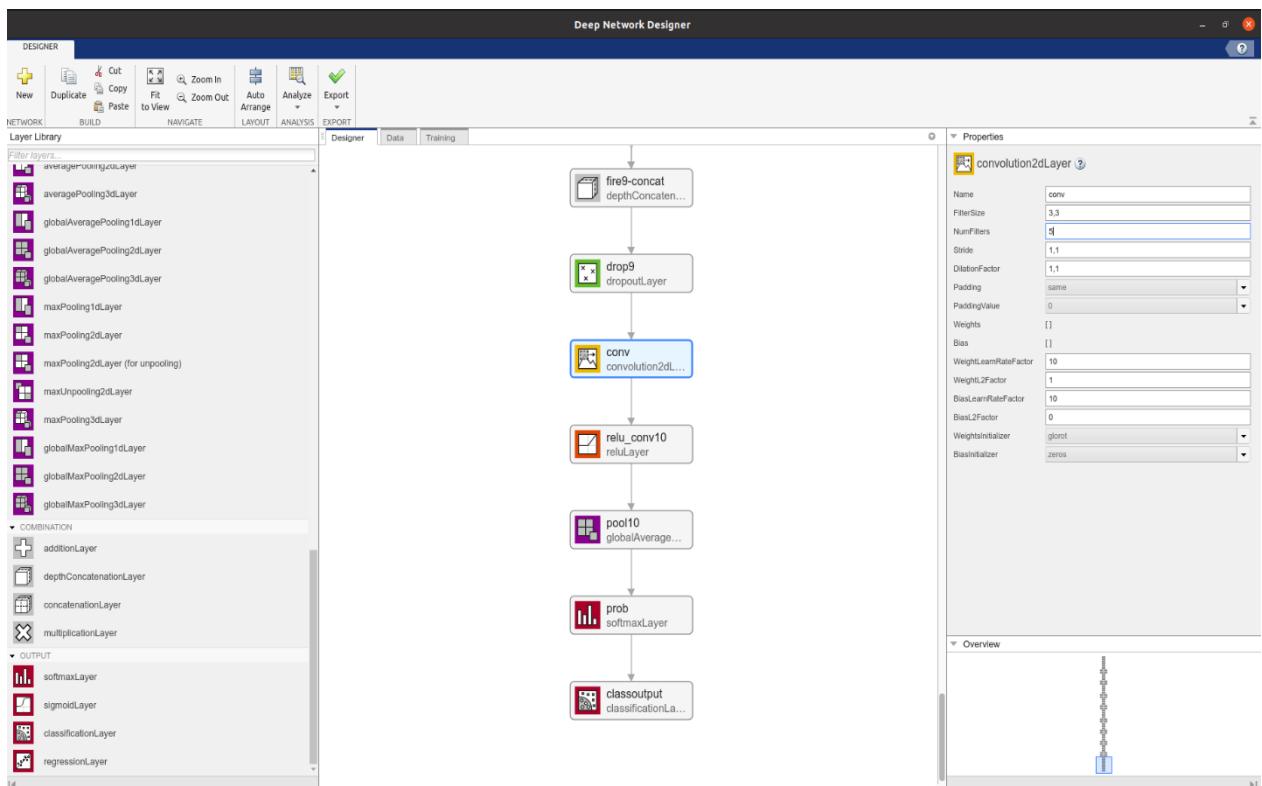
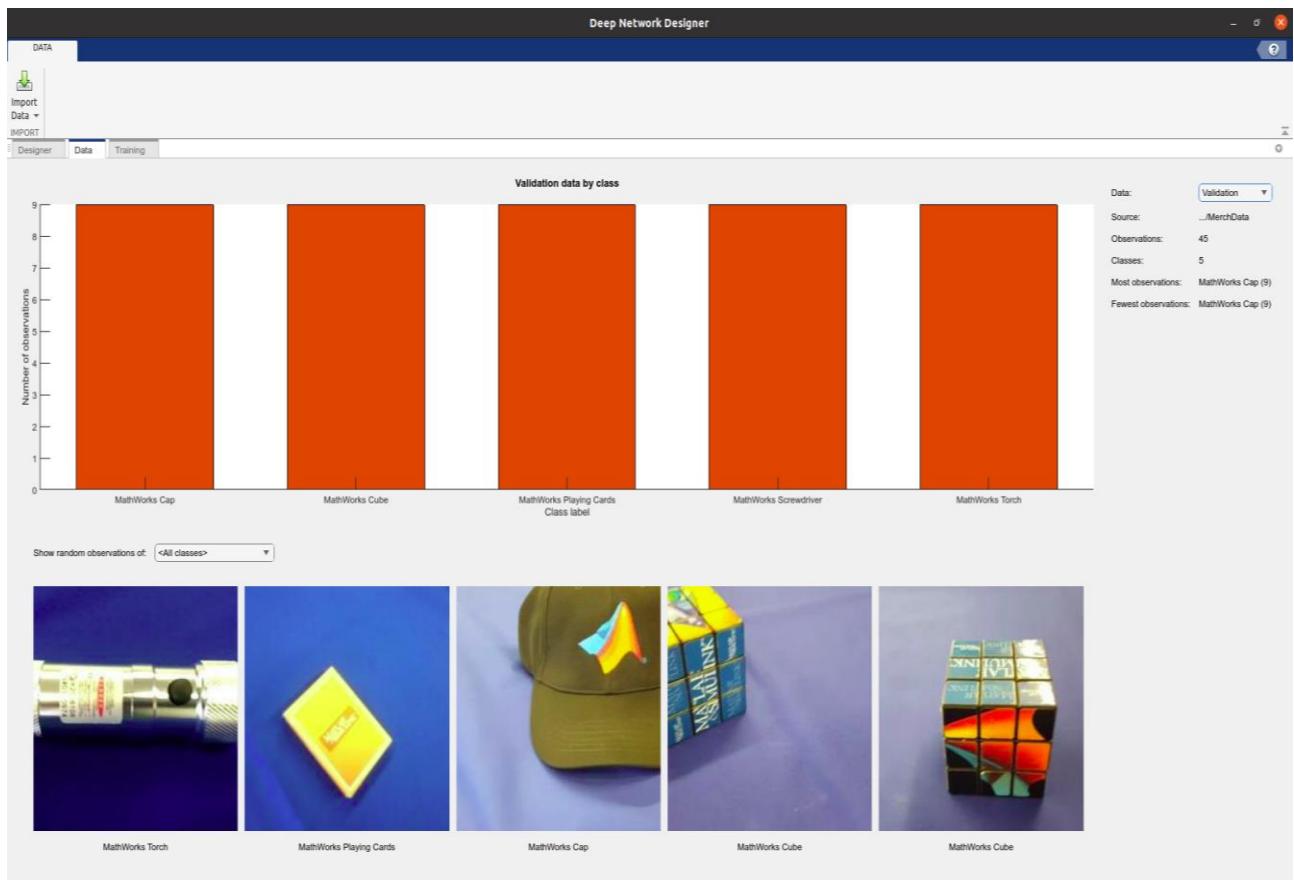
[YPred,probs] = classify(trainedNetwork_1,I);
imshow(I)
label = YPred;
title(string(label) + ", " + num2str(100*max(probs),3) + "%");
```

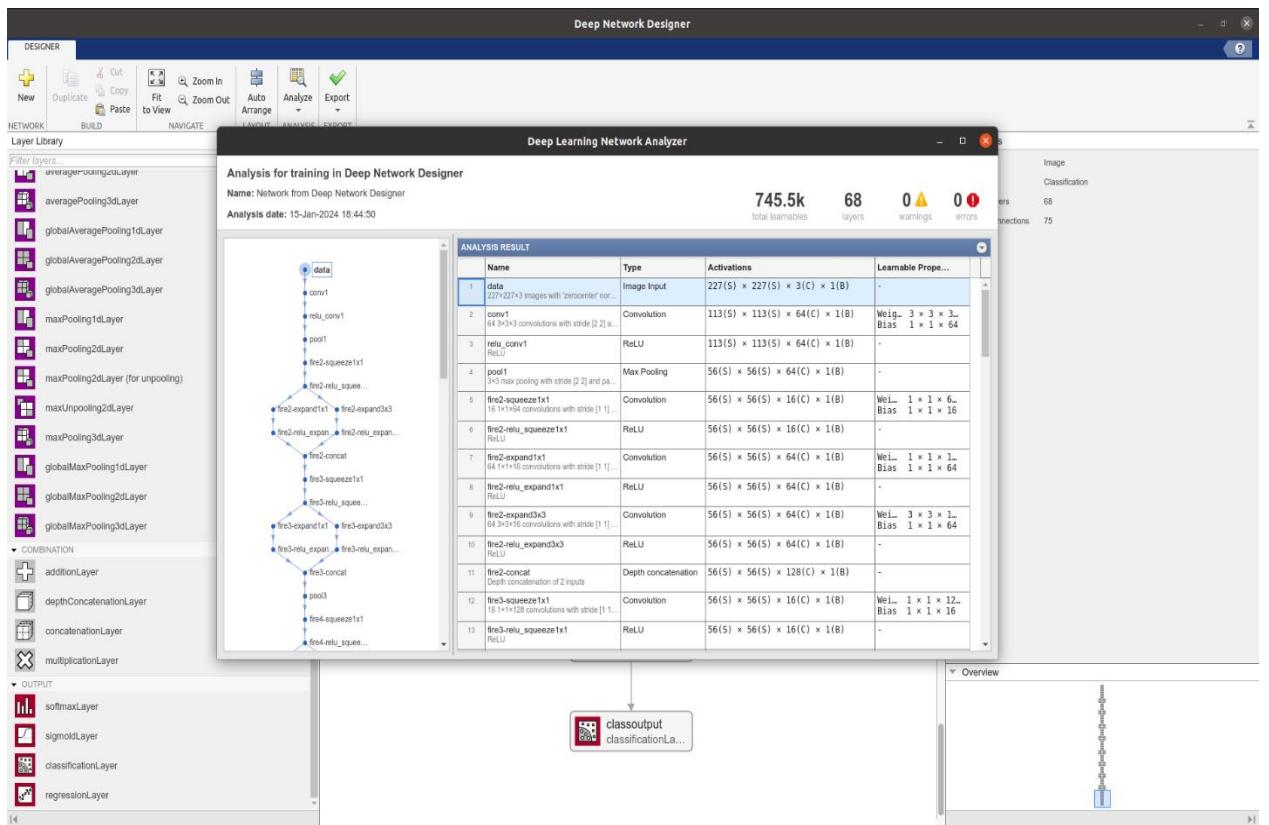
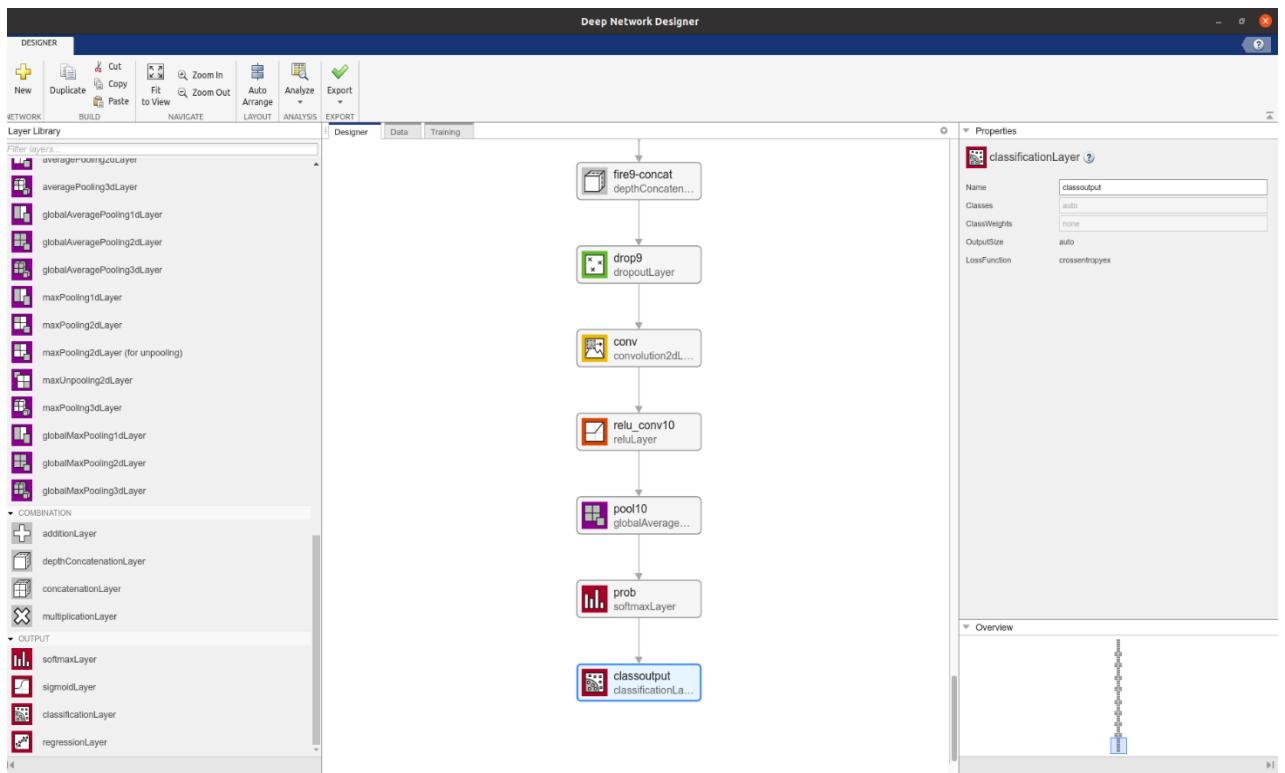
Αποτέλεσμα: Σωστή αναγνώριση(100%) της κλάσης στην οποία η εικόνα MerchDataTest.jpg, δηλαδή στην κλάση MathWorks Cube.

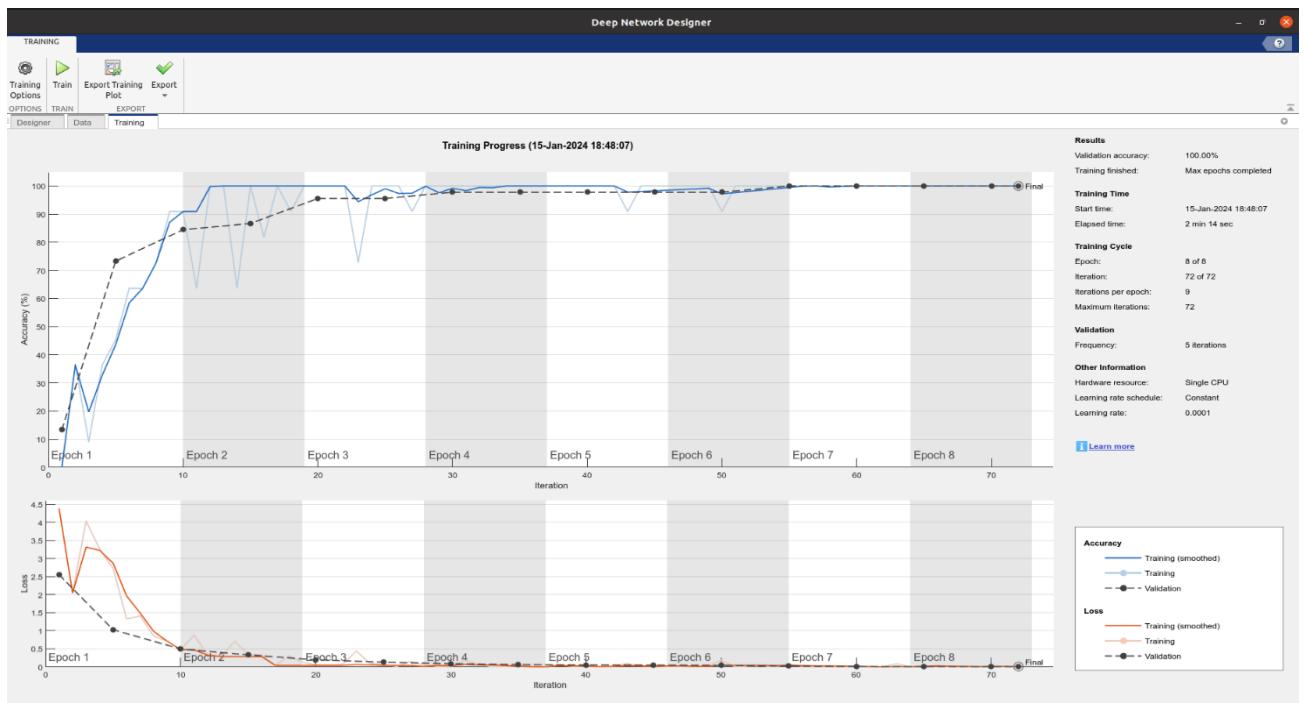
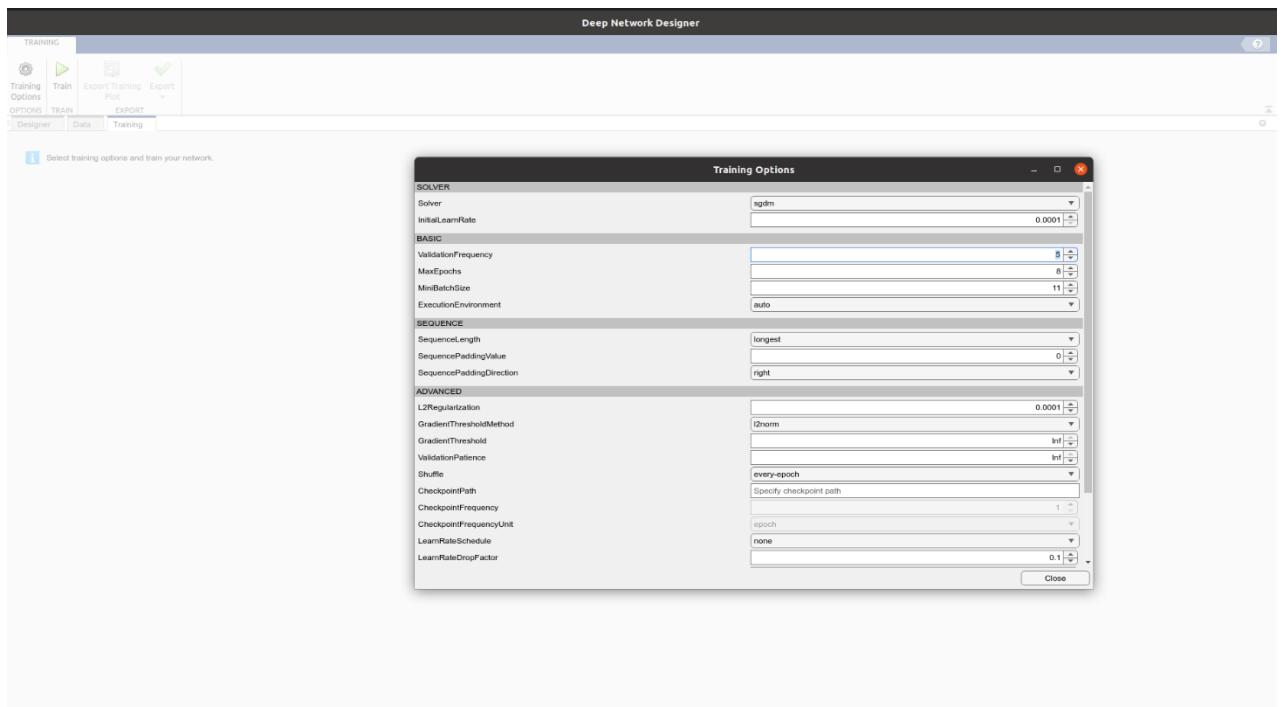


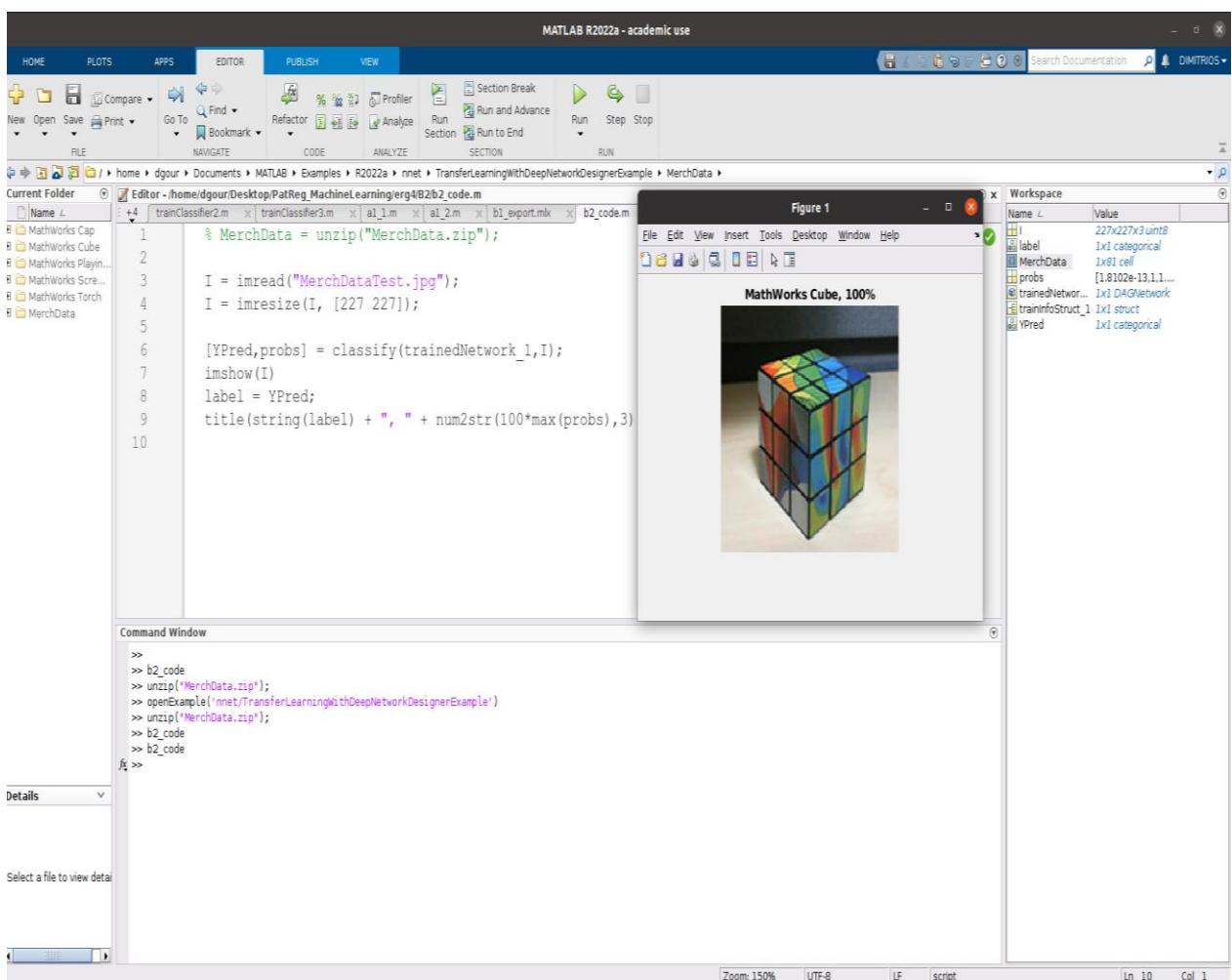
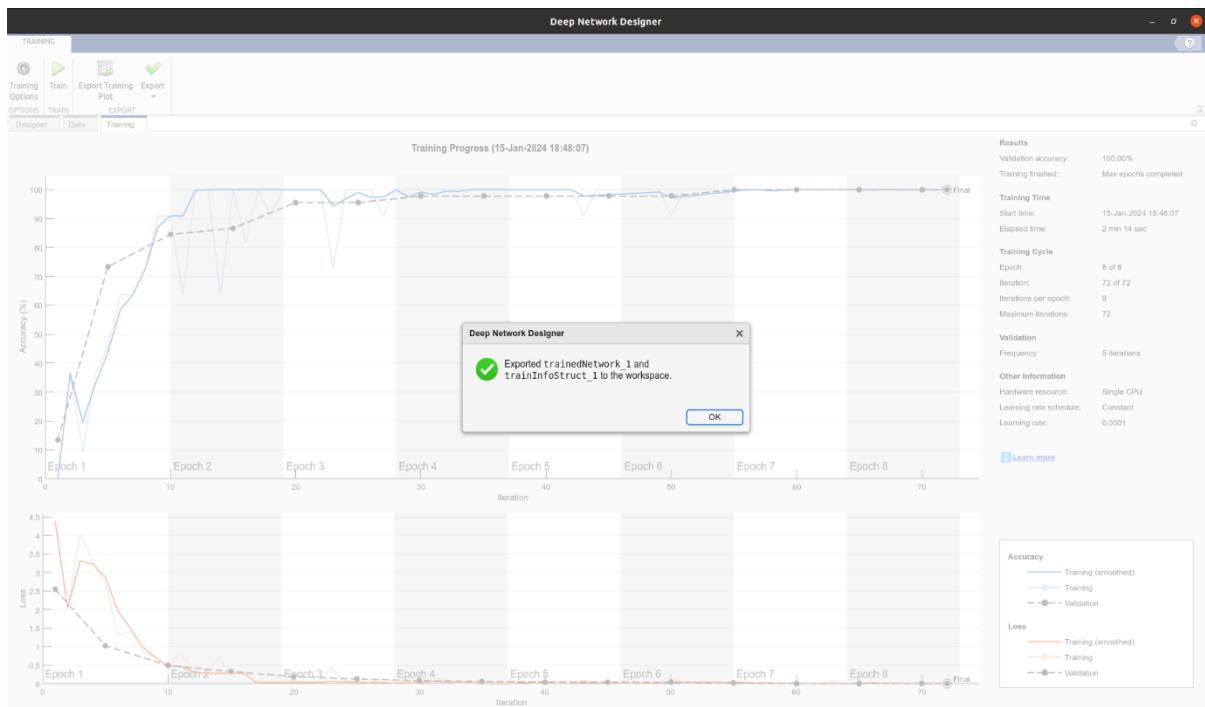












Σχολιασμός αποτελεσμάτων:

Στο συγκεκριμένο παράδειγμα χρησιμοποιούμε το προεκπαιδευμένου νευρωνικό δίκτυο SqueezeNet, που αποτελεί ένα ελαφρύ νευρωνικό δίκτυο που ασχολείται με κατηγοριοποίηση εικόνων, και αλλάζουμε τα τελευταία στρώματα του.

Συγκεκριμένα, όπως παρουσιάζεται και στις παραπάνω εικόνες, αλλάζουμε το τελευταίο στρώμα του δικτύου, αλλά και το τελευταίο conv στρώμα. Οι κλάσεις που υπάρχουν σε αυτό το παράδειγμα είναι 5: Cap,Cube,Playing Cards,Screwdriver,Torch. Στο σύνολο δεδομένων υπάρχουν 105 παρατηρήσεις για εκπαίδευση, δηλαδή 21 από κάθε κλάση. Παράλληλα, για validation υπάρχουν 45 παρατηρήσεις.

Το δίκτυο εκπαιδεύεται, φτάνοντας 100% validation accuracy. Στο διάγραμμα παρατηρούμε την άνοδο της απόδοσης του δικτύου ανά εποχή, όπως και τη μείωση των λανθασμένων κατηγοριοποιήσεων. Τέλος, επιλέγουμε την εικόνα **MerchDataTest.jpg**, που ανήκει στην κλάση Cube. Με τη χρήση της συνάρτησης classify και ως όρισμα το νευρωνικό δίκτυο που δημιουργήσαμε (παραλλαγμένο SqueezeNet) παρατηρούμε ότι η εικόνα κατηγοριοποιήθηκε στην σωστή κλάση.