# EE454 Project 1 Report

Sean Curran Stephen Durofchalk Zachary Feldman
Ryan Wails

September 23, 2015

# a Project Summary

## a.1 Purpose

The purpose of this project was to implement a pre-trained convolutional neural network (CNN) using MATLAB. The CNN is composed of layers - each layer performs an operation on a set of input matrices and produces a corresponding output matrix. When these layers are composed together, they produce a network; the input of this network is an image, and the output of this network is a vector of classification probabilities.

## a.2 Operations

### a.2.1 Image Normalization Layer

Image normalization takes each image and performs a linear scaling operation on each pixel in the image. The result is an image where each pixel (in each channel) takes on a value in the range [-0.5 to 0.5].

```
% returns matrix out - the result of the image normalization
% inputs: in - image to be normalized
%         N and M - dimensions of input image
function out = ImNorm(in, N, M)
in = double(in);
out = double(in);
for i=1:N
    for j=1:M
        for k=1:3
            % normalize
            out(i,j,k) = in(i,j,k)/255.0 - 0.5;
        end
    end
end
```

../../matlab/ImNorm.m

## a.3 Rectified Linear Unit Layer

Rectification takes an input image and thresholds all the negative values to 0. This layer emulates neurons with a particular activation function.

```
% returns matrix out - the result of the rectified linear unit
% inputs: in - input image
%         N,M,D - dimensions of the input image
function out = ReLU(in, N, M, D)
out = double(in);
```

```
for i=1:N
    for j=1:M
        for k=1:D
            % any negative numbers become 0 in the output
            out(i,j,k) = max(in(i,j,k), 0);
        end
    end
end
```

../../matlab/ReLU.m

### a.4 Pooling Layer

Pooling layers reduce variance in the image and reduce the overall size of the image by taking patches of the image and compressing that patch to a single pixel.

```
%out - is the matrix result of Maxpool
%this funciton reduces in from an N X M X D
% to an N/2 X M/2 X D/2
function out = Maxpool(in, N, M, D)
%intialize result
out = zeros(N/2, M/2, D);
out = double(out);
%for each pixel in the result image
    for i = 1:N/2
        for j = 1:M/2
            for k = 1:D
                %take a max of the 4x4 pool
                vector=[in(2*i-1,2*j-1,k) in(2*i-1,2*j,k) in(2*i
    ,2*j-1,k) in(2*i,2*j,k)];
                out(i,j,k) = max(vector);
            end
        end
    end
end
```

../../matlab/Maxpool.m

### a.5 Convolutional Layer

The convolution operation takes a set of filters and bias values determined from the CNN's training phase and applies the over an image. The filter is applied with a convolution operator, and the bias is then added to every resultant pixel.

```
%returns matrix out - the result of the convolution
%in - is the image we are convulting agains
%filterBank - array of filters to convolute in with
%biases - array of biases to add after convolutions
function out = convolve(in,filterBank,biases)

    inDimensions = size(in);
    N = inDimensions(1);
    M = inDimensions(2);

    %convert image to doubles
    im = double(in);

    filterDimensions = size(filterBank);

    %initialize the result of convolution
    out = double(zeros(N,M,filterDimensions(4)));

    %used to add bias to convolution result
    oneMatrix = ones(N,M,1);

    %for each 3d filter in the filterBank
    %run this for loop
    for l=1:filterDimensions(4)
        %get current 3d filter
        lfilter = double(filterBank(:,:,:,l));
        sums = double(zeros(N,M,1));
        %convolute image with 3d filter
        for k=1:inDimensions(3)
            temp = imfilter(im(:,:,k),lfilter(:,:,k),0,'same','conv');
            sums=sums+temp;
        end
        %add in biases
        sums = sums + (biases(l)*oneMatrix);
        out(:,:,l) = sums;
    end
end
```

../../matlab/convolve.m

### a.6 Fully-Connected Layer

The fully-connected layer takes the output from the previous layer and maps it against every neuron in the network. The output is a 1D vector of classifications (realized with soft max).

```
function out = FuCd(img, filter, bias)

img = double(img);
filter = double(filter);

img_size = size(img);
filter_size = size(filter);

n = img_size(1);
m = img_size(2);
d1 = img_size(3);

d2 = filter_size(4);

sum = 0;

for a=1:d2
    sum = 0;
for i=1:n
    for j=1:m
        for k=1:d1
            f_ijk = filter(i,j,k,a);
            i_ijk = img(i,j,k);
            sum = sum + f_ijk * i_ijk;
        end
    end
end
    out(1,1,a) = sum + bias(a);
end
```

../../matlab/FuCd.m

## a.7 Soft Max Layer

The soft max layer takes the output from the fully-connected layer and maps it as input to a probability space. Each probability in this output vector corresponds to a classification likelihood.

```
% returns matric out - the result of the softmax
% inputs: in - input image
%         D - third dimension of the input image
function out = Softmax(in, D)
out = double(in);
% calculate maximum across all values in the input
a = max(in(1,1,:));
sum = 0;
for k = 1:D
    sum = sum+ exp(in(1,1,k)-a);
```

```
11 end
   for k = 1:D
13     % normalize all values to be between 0 and 1 and sum to 1
       out(1,1,k) = exp(in(1,1,k)-a)/sum;
15 end
```

../../matlab/Softmax.m

# b Procedural Approach

# c Experimental Observations

## c.1 Intermediate Output

Intermediate output generated by the CNN is included in the appendix attached to this report. Each figure shows the concatenated output of each layer as a gray scale image. The images can be produced by running 'demo.m.'

Observation of the output from each layer is consistent with the project description. Normalization produces an image with scaled pixel values but is relatively unchanged. Convolution produces a set of images with different feature sets extracted. The ReLu produces an image with a 0 threshold (MATLAB's imagesc command slightly changes the output image because the range of the pixel values in the image changes). Max Pooling reduces the size of the image by a factor of (nxn) and has the effect of making the previous image look more "blocky" via down-sampling. Finally, we have the fully-connected layer and the soft max layer which produce a vector of classification probabilities from the image which are represented by the bar graph.

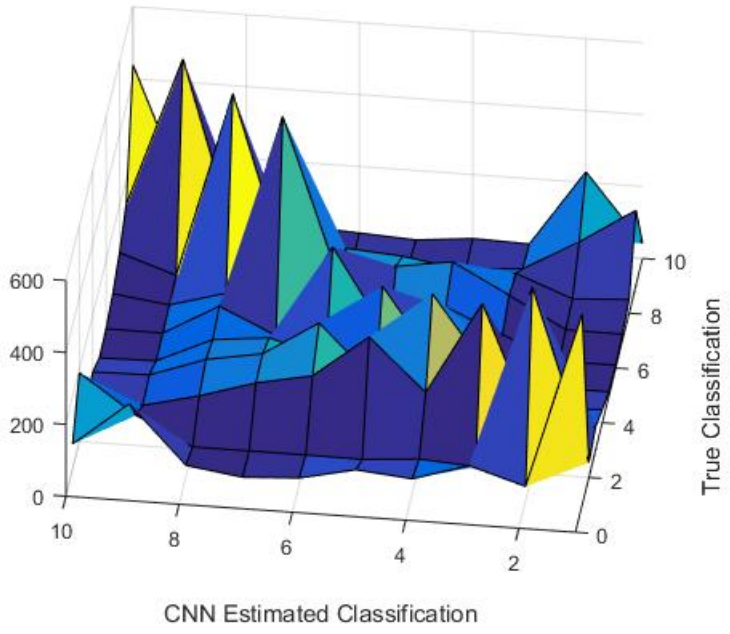
# d Performance Evaluation

## d.1 Results

The following results were the outcome of classifying images as the class associated with the highest probability in the probability vector.

| | CNN Estimated Classification | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| True Classification | | | | | | | | | | | |
| 1 | | 531 | 41 | 65 | 37 | 10 | 8 | 18 | 38 | 210 | 42 |
| 2 | | 40 | 519 | 9 | 26 | 10 | 7 | 19 | 29 | 111 | 230 |
| 3 | | 87 | 8 | 386 | 117 | 97 | 70 | 104 | 88 | 25 | 18 |
| 4 | | 39 | 18 | 127 | 325 | 45 | 136 | 186 | 89 | 13 | 22 |
| 5 | | 53 | 6 | 270 | 69 | 259 | 38 | 162 | 114 | 22 | 7 |
| 6 | | 19 | 7 | 151 | 222 | 49 | 281 | 111 | 125 | 20 | 15 |
| 7 | | 10 | 7 | 120 | 125 | 93 | 23 | 557 | 33 | 9 | 23 |
| 8 | | 32 | 7 | 73 | 98 | 77 | 94 | 54 | 533 | 13 | 19 |
| 9 | | 192 | 84 | 35 | 44 | 7 | 8 | 10 | 16 | 542 | 62 |
| 10 | | 69 | 191 | 23 | 41 | 4 | 9 | 30 | 68 | 127 | 438 |

**Accuracy = 43.7%**



**Discussion:**

The surface plot above demonstrates how well the CNN classifies images. There is a very strong response on the diagonal which represents the correct classifications. What is interesting is that this matrix is somewhat diagonal. This demonstrates that if images in a class of images $C_1$ are being confused for images in a class of images $C_2$, then images in $C_2$ are also being confused for images in $C_1$. A lot of this confusion is entirely logical; for example, trucks and automobiles are frequently confused for each other. Another great example of confusion are cats and dogs are frequently confused for one another. We are again very impressed with how well the CNN because it not only detects similarities between images within a class, but it detects similarities amongst images spanning multiple classes.

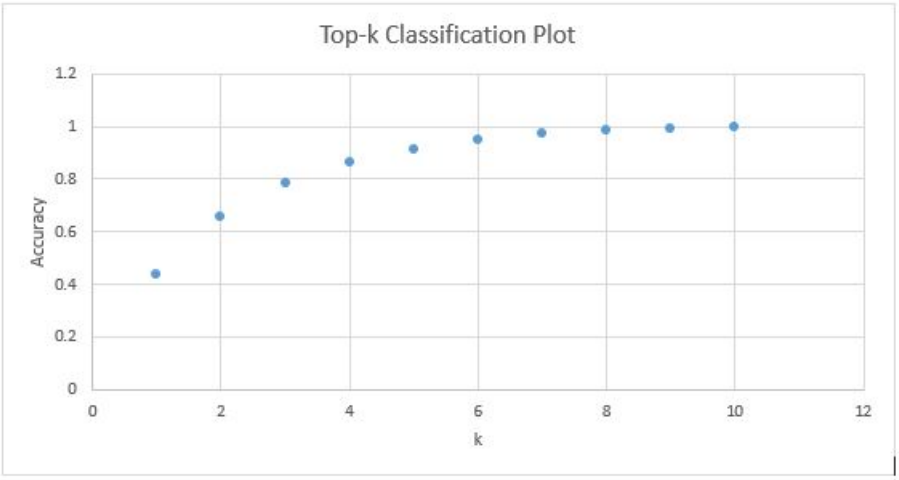
Note: The above results were produced using NeuralNet function located in 'NeuralNet.m'.

## d.2 Looking at the Top-k Highest Probabilities

The following subsection looks at how frequently an image's true class appears in one of the top-k ranked classes, as determined by probability scores sorted from highest to lowest.

| K | Accuracy |
|---|---|
| 1 | 0.4371 |
| 2 | 0.6591 |
| 3 | 0.7864 |
| 4 | 0.8626 |
| 5 | 0.9135 |
| 6 | 0.9496 |
| 7 | 0.971 |
| 8 | 0.9847 |
| 9 | 0.9942 |
| 10 | 1 |

Note: Accuracy is the percentage of times that the correct object class appeared in the top-k ranked classes, as determined by probability scores sorted form highest to lowest.




**Discussion:** We were very surprised to see the top-k classification plot follow an exponential distribution. This demonstrates that the probability estimations provided by the CNN are accurate. We expected the probability estimations to be an arbitrary metric, but looking at the logarithmic nature of the plot it is clear that the probabilities are very accurate.

Note: The above results were produced using the topk function located in 'topk.m' and NeuralNet function located in 'NeuralNet.m'.

# e Further Exploration

## e.1 Test Cases

To test the robustness/accuracy of the training outside of the given data set, additional test images were gathered and fed into the CNN. Images were rescaled to thumbnail size (32x32x3) with the Matlab command

```
thumbnail = imresize(img, [32 32]);
```

110 images total were collected; 100 fell into the preexisting image classes, 10 contained scenes not falling into any preexisting class. Running these new 10 images through the CNN yielded the following results:

| Image # | Image Contents | CNN Classification |
|---|---|---|
| Image 1 | Tree in Field | Horse (8) |
| Image 2 | Winter Mountain Scene | Ship (9) |
| Image 3 | Ryan and Zach | Bird (3) |
| Image 4 | Steve | Frog (7) |
| Image 5 | Sean | Cat (4) |
| Image 6 | Desktop Computer | Automobile (2) |
| Image 7 | Football on Field | Deer(5) |
| Image 8 | Robert Collins | Truck (10) |
| Image 9 | Old Main | Bird(3) |
| Image 10 | Penn State Logo | Truck (10) |

All of the exploratory images we used were either taken by us or were taken from Google Images. The code we used to read in the exploratory images can be found in 'getAllFiles.m' and 'getTrueClass.m'. We produced the results above by running the 10 images through the function found in 'NeuralNet.m' and classifying the images by assigning the image the class associated with the highest probability. The classification process was done manually.

## e.2 Reclassification

An attempt was made to distinguish these 10 images from the rest of the data set (i.e. reclassify these 10 images as unknown). Let $V$ be the vector of output probabilities from the CNN for each image. Let $C$ be the classification of the image producing probability vector $V$. Originally, we have

$p_i \in V$
$p_{max} = \max(p_1, p_2, ..., p_{10})$
$C = i$ given $p_i = p_{max}$

To reclassify the images we use
$p_{max} = \max(p_1, p_2, ..., p_{10})$
$P = \{p_1, p_2, ..., p_{10}\} \setminus \{p_{max}\}$
$p_{2max} = \max(P)$

$$\begin{cases} C = i \text{ given } p_i = p_{max} & \text{for } p_{max} - p_{2max} >= 0.25 \\ C = 11 & \text{otherwise} \end{cases}$$

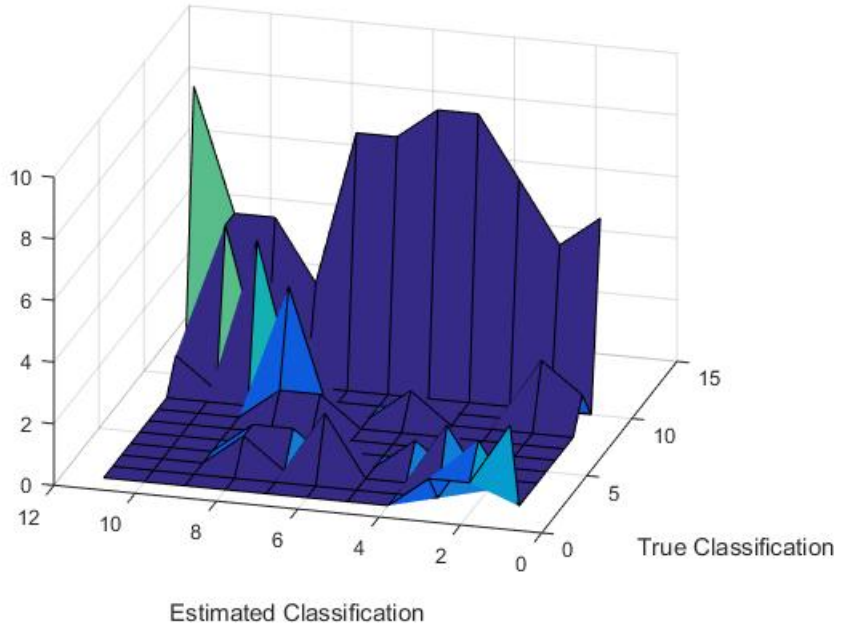
Less precisely, the difference between the two peak responses was thresholded by .25; so, if there were two strong responses, the image is reclassified as unknown.

The code used to reclassify images can be found in 'identify_exploratory.m'.

### e.3 Results After Reclassification

In the confusion matrix below, classes 1 through 10 are the same classifications. Class 11 identifies an unknown image class.

| | CNN Class | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| True Class | | | | | | | | | | | | |
| 1 | | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 6 |
| 2 | | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 5 |
| 3 | | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 |
| 4 | | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 9 |
| 5 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 9 |
| 6 | | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 |
| 7 | | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 8 |
| 8 | | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 4 | 0 | 0 | 3 |
| 9 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 5 |
| 10 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 5 |
| 11 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 9 |




**Accuracy = 29.09%**

The reclassification metric does well in replacing the 10 new images in the unknown category, but also places images that were previously correctly classified in the unknown category. Other metrics tested were

1. Thresholding on the number of classes that had a response above 0.1 (the mean value for random classification)

2. Taking the spatial derivative of the probability vector; this metric actually produces zero-mean Gaussian noise with a very small standard deviation.

The code used to produce the results found in this section can be found in 'exploratory_main.m'.

## f Member Contributions

Overall the contributions made form each team member were pretty even. Whenever the team met to work on the project most of the team members were in attendance. Everyone who was in attendance at each team meeting attempted to work on a different piece of the project to ensure that we were making steady progress. The coding was not split up evenly as one team member did not attend the initial meeting and as a result he missed most of the actual coding and basically all of the design decisions. We attempted to split the report up evenly, but as the deadline approached scheduling meetings became difficult and some individuals were unavailable to work on the report. In conclusion, the work was not split up evenly among our four group members, but everyone attempted to contribute.

# Appendices

## A Intermediate Output

Figure 1: Original Image

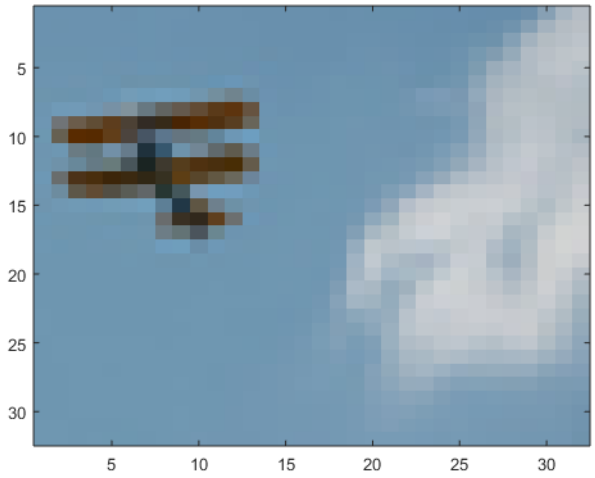


Figure 2: Layer 1 Output

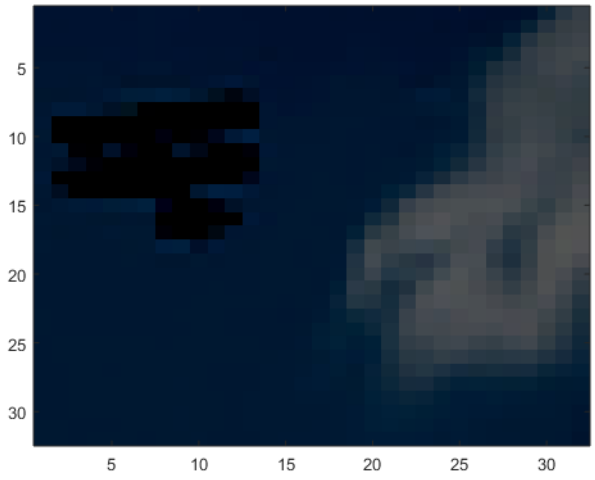



Figure 3: Layer 2 Output

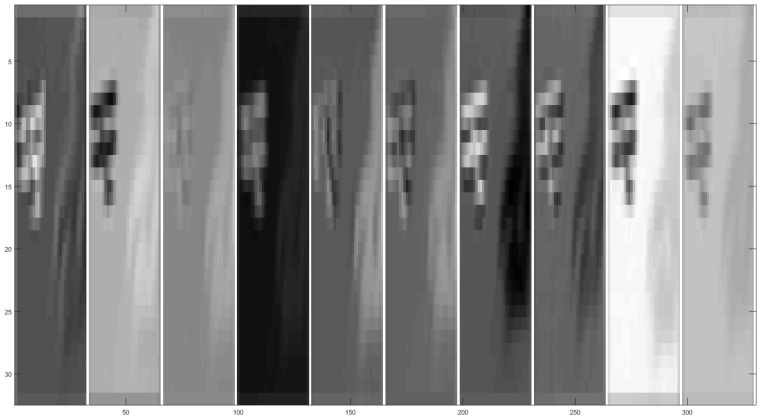


Figure 4: Layer 3 Output

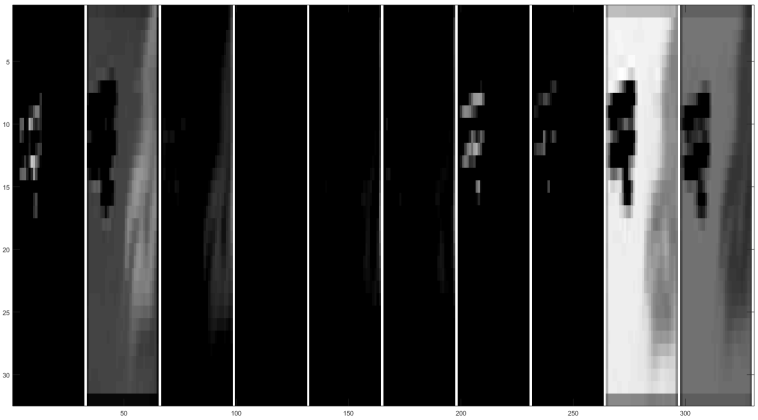



Figure 5: Layer 4 Output

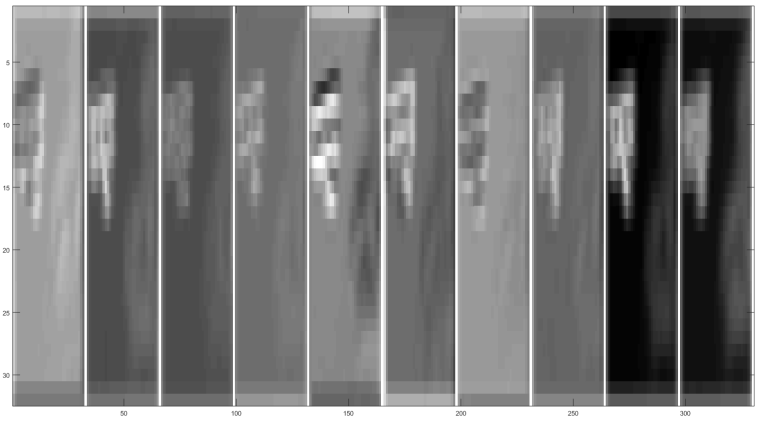


Figure 6: Layer 5 Output

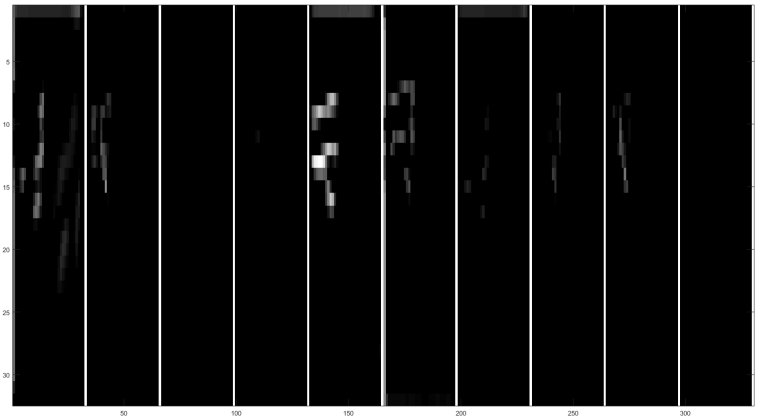



Figure 7: Layer 6 Output

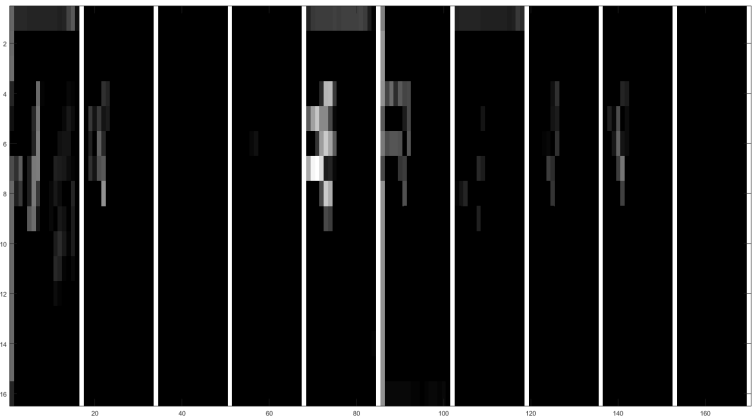


Figure 8: Layer 7 Output

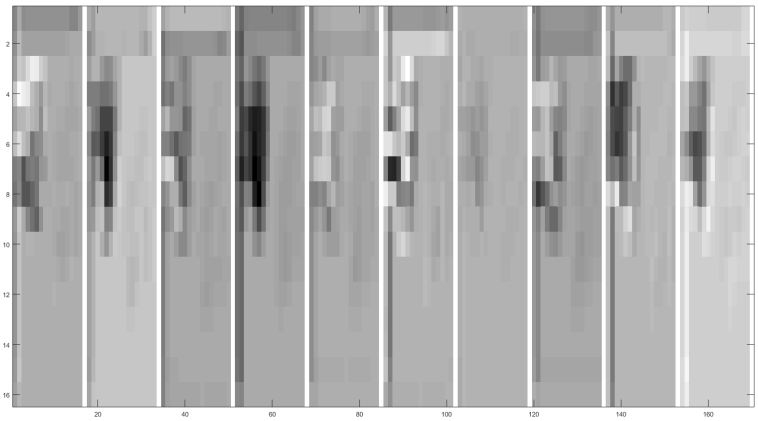



Figure 9: Layer 8 Output

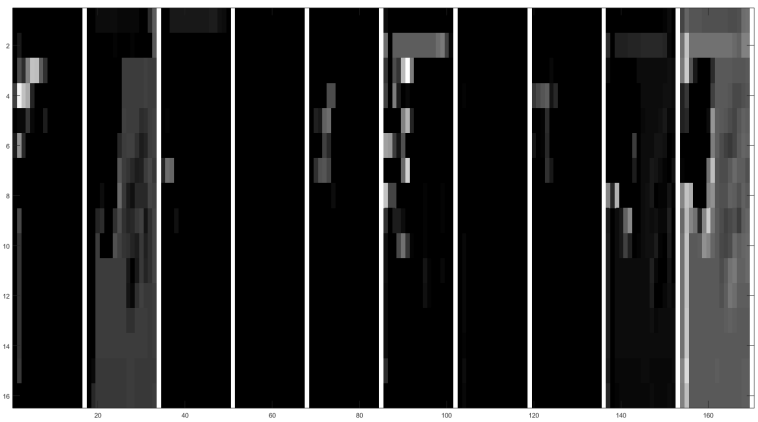


Figure 10: Layer 9 Output

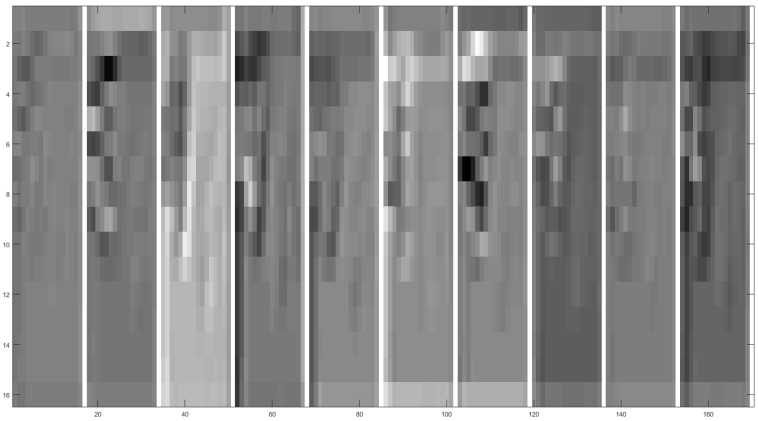



Figure 11: Layer 10 Output

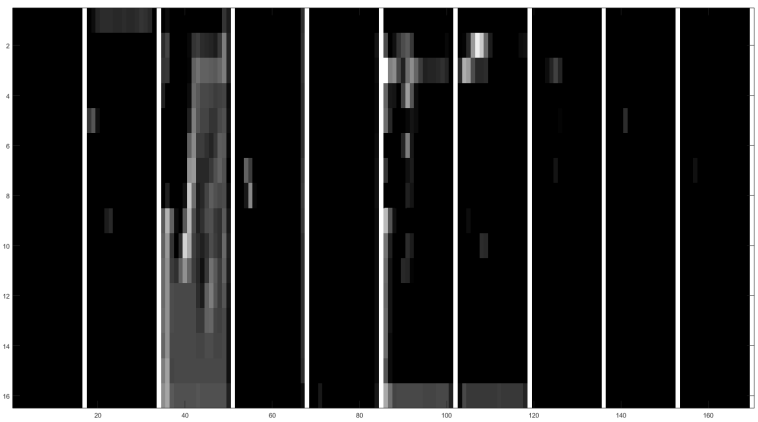


Figure 12: Layer 11 Output

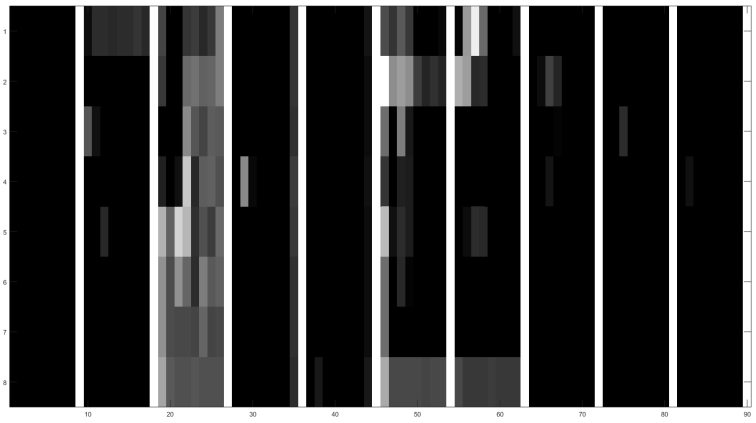



Figure 13: Layer 12 Output

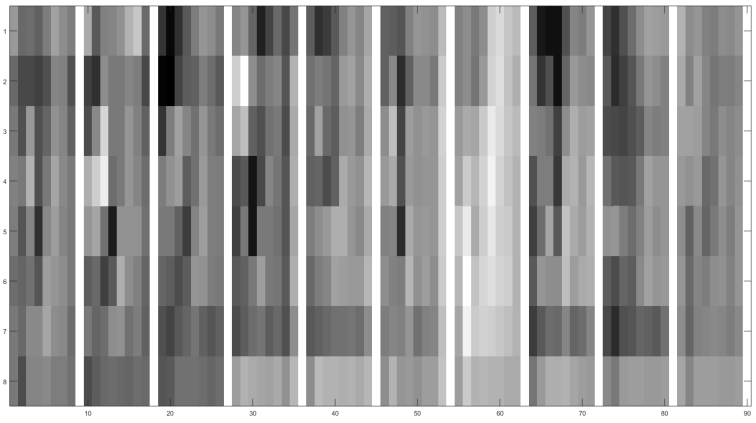


Figure 14: Layer 13 Output

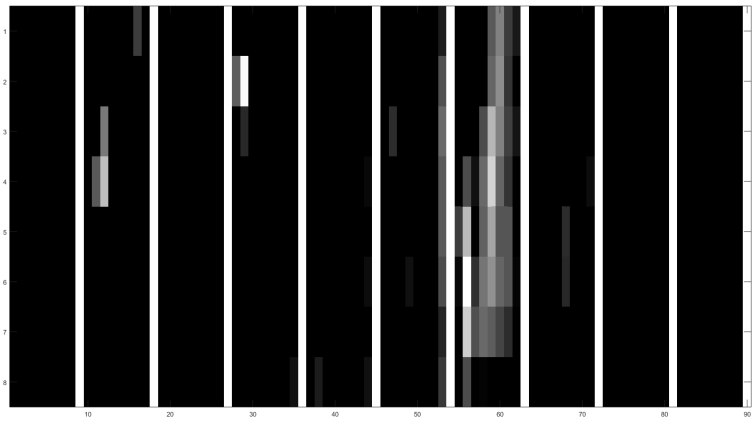



Figure 15: Layer 14 Output

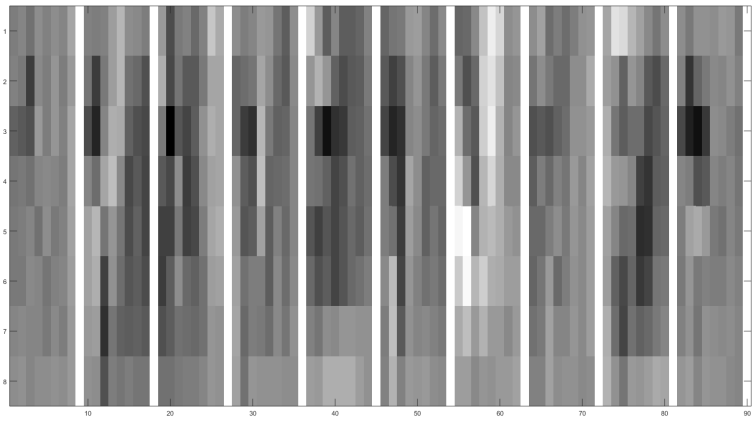


Figure 16: Layer 15 Output

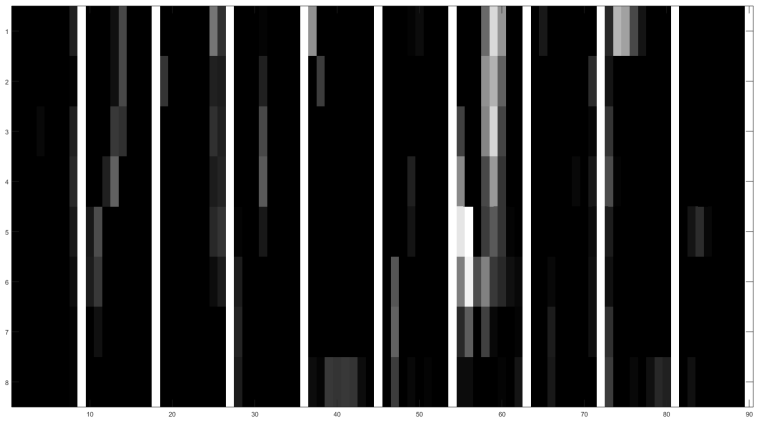



Figure 17: Layer 16 Output

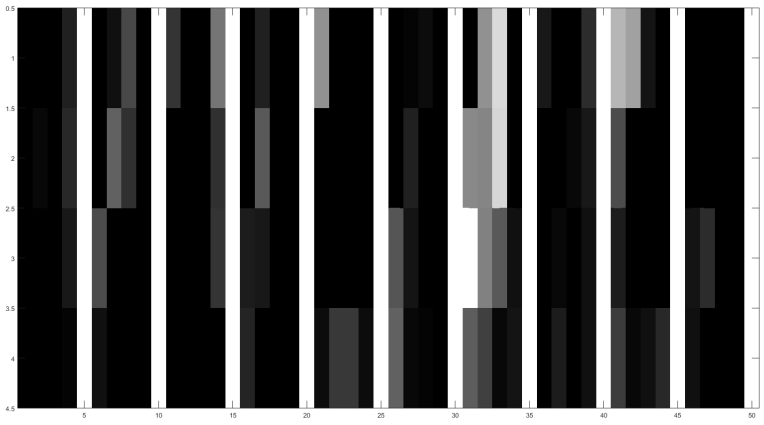


Figure 18: Layer 17 Output

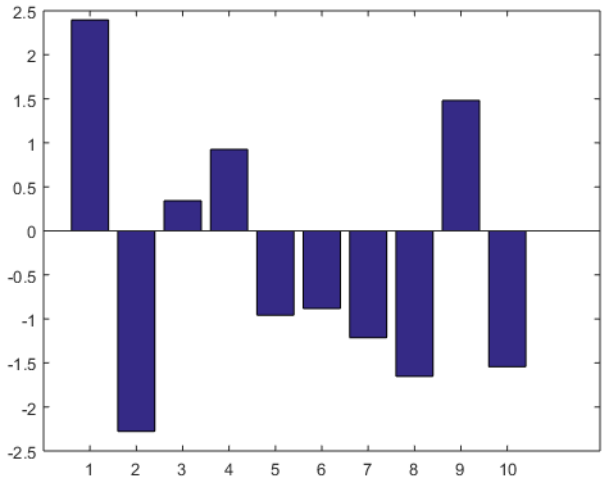



Figure 19: Layer 18 Output