Homework 0 - Alohomora!
USING 5 LATE DAYS

John Kanu
Email: john.d.kanu@gmail.com

***Abstract*—In this paper I describe my implementation of PbLite and neural networks for computer vision.**

I. Phase 1: Shake My Boundary

In this phase, we implemented a range of features that are correlated with boundaries in an image. More precisely, we implemented operations which map an image I to a 2-dimensional image J where the magnitude of J_{xy} is proportional to the strength of the gradient in the original image I at pixel (x, y), i.e. the probability of boundary in I.

Combining the features together, we were able to implement a fairly well-performing algorithm for edge detection, which outperforms the classic Sobel and Canny baseline. By incorporating brightness, color, and texture gradient information with hundreds of filters at varying scale and orientation, the Pb-lite is able to capture a wide range of information that is left out by any single component, as evident in the Sobel and Canny baseline. By expressing the probability of boundary as a linear combination with weights summing to 1, the computation of the probability by Pb-lite effectively considers weighted votes by each of the texton gradient, brightness gradient, color gradient, CannyPb, and SobelPb (the latter two with variable weights), each of which is able to detect boundaries of a particular type, but none of which is able to detect boundaries of all types.

A. Oriented DoG filters

The first generated filter was the oriented derivative of gradient (DoG) filter. This task involved generating a filter bank of oriented Derivative of Gaussian (DoG) filters.

1) Implementation: The filter bank consists a set of filters over the two dimensions of size and orientation. Orientations are defined in the range $[0, 2\pi)$, containing no duplicates, and size is given as some arbitrary odd number, defining a valid kernel for convolution. See Fig 1. for a sample filter bank.

Fig. 1. Sample oriented DoG filter bank

There are several methods for generating the filters, including the direct computation of the derivative of the Gaussian, or convolving a simple Sobel filter and a Gaussian kernel, which is an approximation of the actual value. For simplicity, I employed the latter.

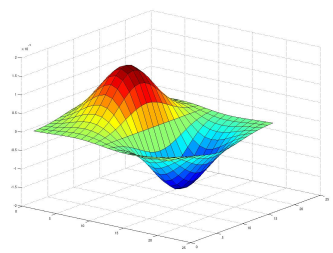

Fig. 2. Derivative of the Gaussian function

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

The Gaussian function is defined in equation (1). The derivative of the Gaussian is visualized in Fig 2, which illustrates the similarity between DoG kernel intensity and the actual value of the derivative of the Gaussian.

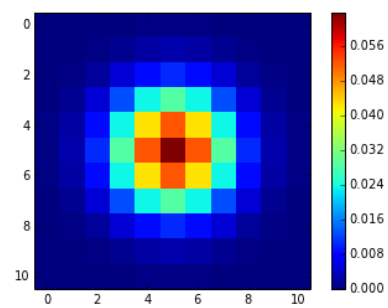

Fig. 3. Gaussian filter

The Sobel operator is defined by two filters, one for the x component and one for the y component, given as G_x and G_y, respectively. Each of these filters is designed for edge detection, and is maximized by a gradient along the axis perpendicular to the row or column of zeros.

Given these definitions, the similarity between the true derivative of Gaussian and the convolution of Sobel and Gaussian filters becomes apparent. Given the positive, zero, and negative entries whose magnitudes are inversely proportional to the distance from center, the convolution of the Sobel operator allows us to approximate the true value. Procedurally, in order to generate the DoG filter for any given orientation, we simply take G_x and convolve it with the Gaussian kernel. Below is the output of this process.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Fig. 4. Sobel filters for each component

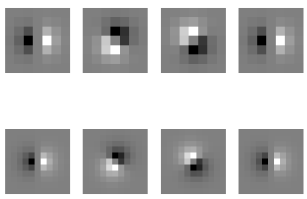

Fig. 5. DoG filters generated for 2 scales and 4 orientations

2) Interpretation and Analysis: Below is an example image from BSDS500. We can clearly see edges between the penguin and the background, between rocks and neighboring rocks, between the penguin's eyes and its face, and between the wide, curved ground in the background and the lighter sky.

Fig. 6. Image 7.png in BSDS500

Below is the image convolved with one of the DoG filters. We can see high intensity around the penguin's edges, and between rocks. However, we do not observe high intensity in the background curve, where there should be an edge. This is due to the orientation of the filter, which picks up vertical edges more than horizontal edges.

B. Leung-Malik Filters

In this section we implemented two filter banks consisting of Leung-Malik filters at two different scales. The filters are designed to detect edges, bars, and spots at multiple scales and orientations, allowing us to detect boundaries, which are often characterized by one or more of these phenomena in the image.

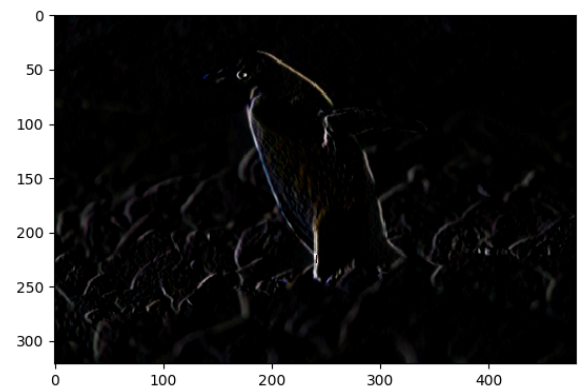

Fig. 7. Sample DoG convolution

1) Implementation: Each filter bank is a set of multi scale, multi orientation filter bank with 48 filters, including first and second order derivatives of Gaussians at 6 orientations and 3 scales; 8 Laplacian of Gaussian (LOG) filters; and 4 Gaussians.

In LM Small (LMS), the filters occur at basic scales $\sigma = \{1, \sqrt{2}, 2, 2\sqrt{2}\}$. The first and second derivative filters occur at the first three scales with an elongation factor of 3, i.e., ($\sigma_y = \sigma$ and $\sigma_y = 3\sigma_x$). The Gaussians occur at the four basic scales while the 8 LOG filters occur at σ and 3σ. Below is a sample output of this process.

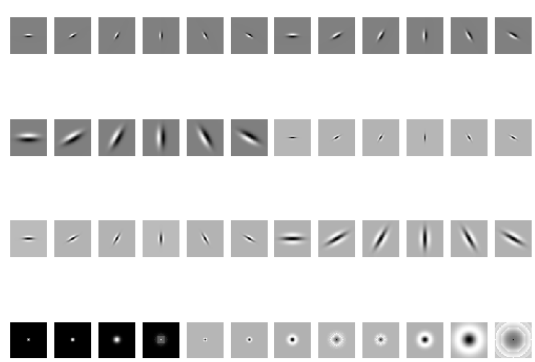

Fig. 8. Leung-Malik Small Filter Bank

For LM Large (LML), the filters occur at the basic scales $\sigma = \{\sqrt{2}, 2, 2\sqrt{2}, 4\}$.

2) Interpretation and Analysis: Figure 10 contains the output of convolution of one of the small (LMS) filters with

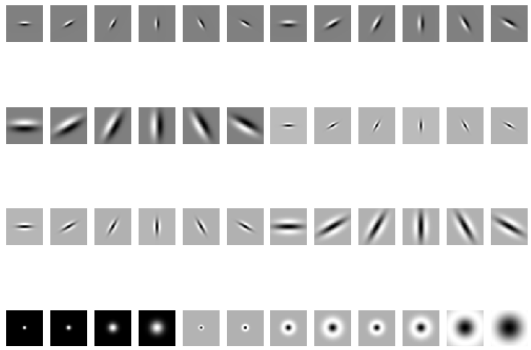

Fig. 9. Leung-Malik Large Filter Bank

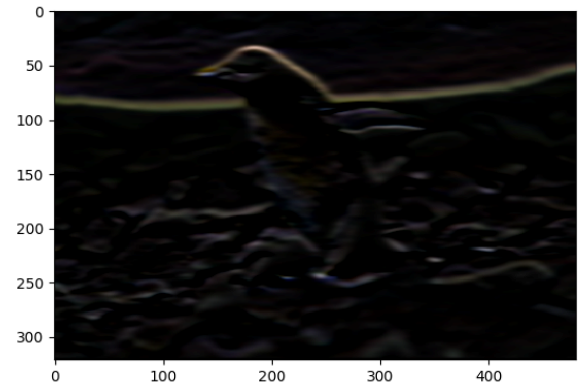

Fig. 11. Sample LML convolution

the same image from BSDS500. Unlike the previous DoG kernel which did not pick up the edge in the background, this particular LMS kernel is oriented so as to detect the edge. Figure 11 demonstrates the convolution operation using large (LML) filters. The output is more blurred than LMS for some of the edges, and more prominent for others, as each kernel operation has a wider receptive field, due to a larger scale. As a result, the filters of different sizes pick up on edges of varying width, allowing the full filter bank to detect a wide range of edges in the image.

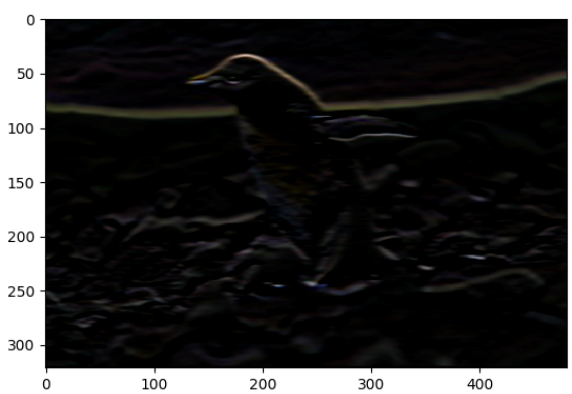

Fig. 10. Sample LMS convolution

C. Gabor Filters

The Gabor Filter is a linear filter used for texture analysis. Unlike the previous two filters, which have a maximum of 2 peaks, the Gabor filter contains many peaks, which allows a convolution of the Gabor filter to pick up higher frequency changes in the surface of the image. Similar to the previous filter banks, I generated multi-orientation and multi-scale Gabor filters.

1) Implementation: The Gabor filter is a Gaussian kernel function modulated by a sinusoidal plane wave, given by the equation below, which defines the value of the filter at given index, orientation, and scale. After implementing this procedure, a filter bank can be generated as in Fig 11.

$$G_c[i,j] = Be^{-\frac{(i^2+j^2)}{2\sigma^2}} \cos(2\pi f(i\cos\theta + j\sin\theta))$$
$$G_s[i,j] = Ce^{-\frac{(i^2+j^2)}{2\sigma^2}} \sin(2\pi f(i\cos\theta + j\sin\theta))$$

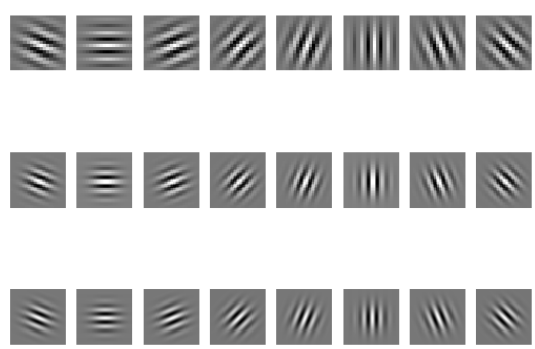

Fig. 12. Gabor filters generated at 3 different scales and 8 orientations

2) Interpretation and Analysis: Figure 13 illustrates a sample output of a Gabor convolution of `7.png` in BSDS500.

Similar to the other filters we've seen so far, this output contains high magnitude around the boundaries. However, the image convolved with the high frequency filter demonstrates a different activation pattern. Qualitatively, there are waves present in the output, which are maximized at regions of the image closely matching the waves in the filter. As we begin to see higher activation slightly within the boundaries of objects in the image, it appears that this filter incorporates texture information into the computation of its output, such that boundaries with higher texture variation yield higher activation.

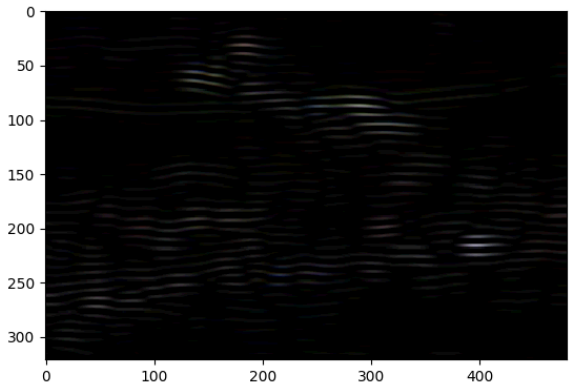

Fig. 13. Sample Gabor convolution

D. Texton Map

Using all of the previously defined filters, we can assign many different attributes to any point in the image. Moreover, these attributes are often not equal, though each one of them represents a valuable piece of information that can be used to predict the probability of boundary. Each feature is limited to a size of 1-dimension (only 1 real number output per pixel), but the features can be combined together to produce a high-dimensional feature vector representing activations of filters of different type, scale, and orientation. This gives a more complete representation of the pixel with respect to boundary information. To produce the map of an image, we cluster the feature vectors and assign each point to a cluster based on the cluster centroids.

1) Implementation: We use the K-means clustering algorithm to cluster the datapoints to produce textons. This is a fairly straightforward operation. There are several hyper-parameters in the K-means algorithm, including the number of clusters k and the initialization of cluster centroids. For simplicity, I run K-means once with $k = 8$.

2) Interpretation and Analysis: As we can see in Figure 14, each pixel is assigned a single texton, illustrated using a unique color. regions of the image that have similar edge and texture patterns are assigned to the same cluster. This allows us to have a much lower dimensional representation of each pixel, which translates into a lower dimensional representation of the entire image, as the image appears to be subdivided into closed regions of uniform texton identification. Boundaries between these regions are evident in the sharp change between one uniform region on one side of the boundary and a different uniform region on the other side of the boundary, with different texton IDs.

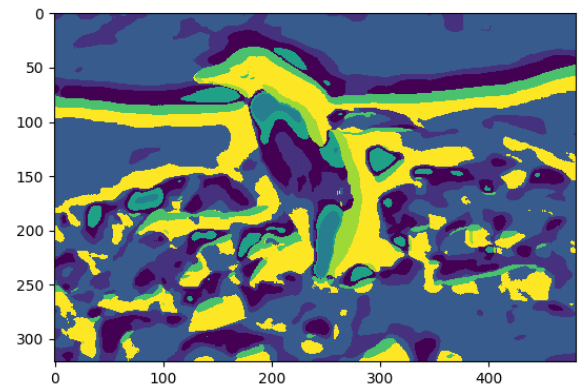

Fig. 14. Texton map

E. Brightness Map

This map operates on the brightness of the pixel, as opposed to the texton. Clustering is performed on the brightness as measured by the intensity of the grayscale image.

1) Implementation: I employ K-means clustering with $k = 8$.

2) Interpretation and Analysis: Figure 15 contains the brightness map of the image. It is apparent that neighboring pixels belonging to the same region are not as cleanly separated as in the texton map. The resulting brightness map contains more noise than the texton map. However, the clusters still have utilitary value in detecting boundaries.

F. Color Map

This map operates on the color of the pixel. Clustering is performed on the RGB color space, as measured by the intensities of the 3 RGB channels of the image.

1) Implementation: I employ K-means clustering with $k = 8$.

2) Interpretation and Analysis: Figure 16 contains the color map of the image. It is apparent that neighboring pixels belonging to the same region are not as cleanly separated as in the texton map. However, the color map is very similar to the brightness map. The resulting color map contains more noise than the texton map. However, the clusters still have utilitary value in detecting boundaries.

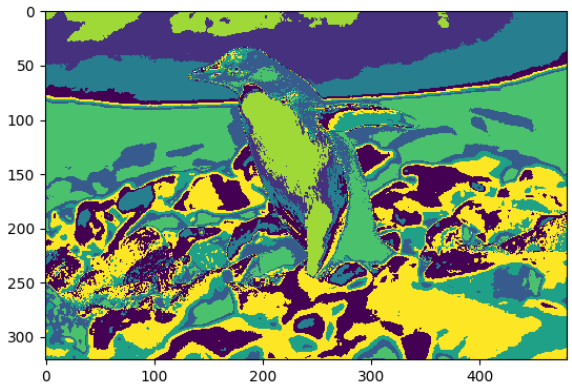

Fig. 15. Brightness map

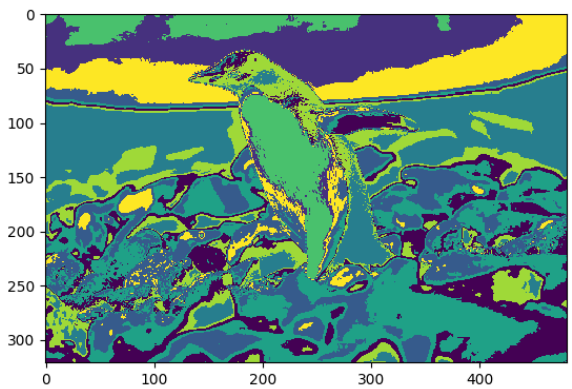

Fig. 16. Brightness map

G. Half-disk masks

Half-disk masks are used to extract image in one direction with respect to a single pixel. Pairs of half-disk masks at opposing directions are combined to compare the information in both opposing directions, to compute a gradient.

1) Implementation: Half-disk masks are generated by selecting the pixels that lie within a circle and inside a half of the circle given by the orientation. Below is the result for varying scales and orientations. The code supports the generation of a pair of masks facing in opposite directions, i.e. with orientations θ and $\theta + \pi$.

H. Texture Gradient

Here we used the computed the gradient in the texton map, which is correlated with probability of boundary.

1) Implementation: The code for computing χ^2 follows the same structure as defined in the specifications on the course website. Division by 0 is avoided by adding a small epsilon value to the denominator in the summation.

Fig. 17. Half-disk masks

2) Interpretation and Analysis: The figures below illustrate Texton gradient for Image 7. The quality of the edges is dependent on the choice of k in the clustering procedure, as well as the choice of scale and orientation of each of the filters described above. The areas of high gradient correspond to the changes in the intensity of the color map.

By visual inspection one can observe that there are many continuous lines of high magnitude. Some boundaries appear in places which a human would not consider a boundary to exist, such as on both sides of the boundary in the background. This is most likely due to the incorporations of large-scale patterns in the high-dimensional feature vector, leading to different textons around the edges. Nonetheless, one can inspect the original image and observe that there slight changes in the image around the boundary, which may be reflected in the gradient. In other contexts, gradients of this kind may actually correspond to boundaries, such as around clouds in the sky. However, more information is needed to rule out the boundary here, which is provided by the brightness gradient and color gradient.

I. Brightness Gradient

Here we used the computed the gradient in the brightness map, which is correlated with probability of boundary.

1) Implementation: The code for computing χ^2 follows the same structure as defined in the specifications on the course website. Division by 0 is avoided by adding a small epsilon value to the denominator in the summation.

2) Interpretation and Analysis: These boundaries are much cleaner than for the texton gradient. However there are regions of gradually changing magnitude within the penguin, which should not be identified as boundaries.

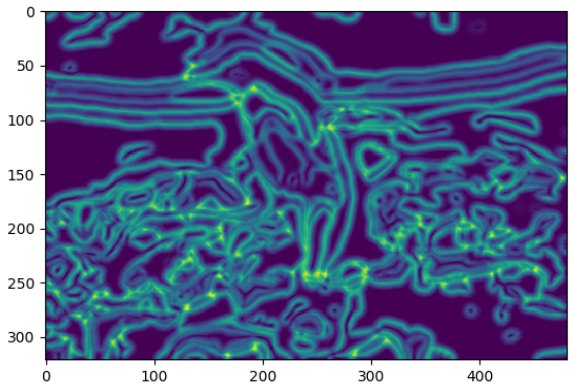

Fig. 18. Texton gradient

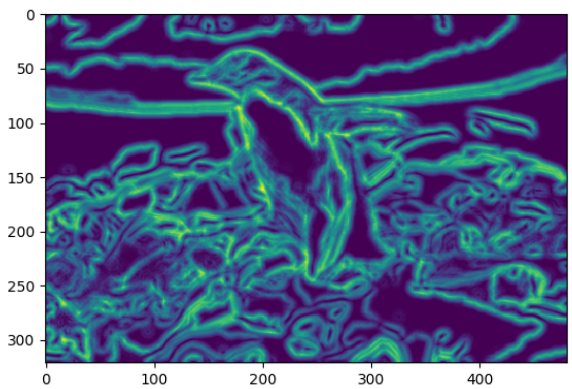

Fig. 19. Brightness gradient

J. Color Gradient

Here we used the computed the gradient in the color map, which is correlated with probability of boundary.

1) Implementation: The code for computing χ^2 follows the same structure as defined in the specifications on the course website. Division by 0 is avoided by adding a small epsilon value to the denominator in the summation.

2) Interpretation and Analysis: The color gradient looks very similar to the brightness gradient. This is likely due to the fact that gradients in color space are not reflected in grayscale if brightness is approximately constant along the gradient. One can observe a different pattern in the sky resulting from the difference in information.

K. Sobel baseline

Figure 21 shows convolution of Image 7 with the Sobel filter

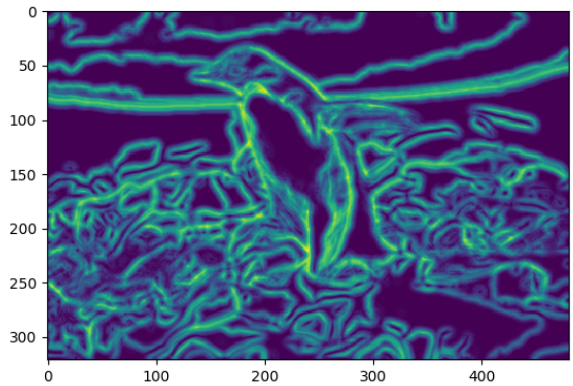

Fig. 20. Color gradient

1) Interpretation and Analysis: Notice that the Sobel convolution fails to detect the boundaries in the background of the image. The distribution of probabilities lies in high density around the penguin and the middle ground in front of and behind the penguin. The Sobel operator has ill-suited to detect the softer boundary in the background, due to the simple definition, which is sensitive only to a subset of all edges that can appear in photos, namely those that are clearly and sharply defined.

Fig. 21. Sobel edges of Image 7

L. Canny baseline

Figure 22 shows convolution of Image 7 with the Canny filter

1) Interpretation and Analysis: In this image, we are able to detect the boundary in the background. However, the areas of high intensity in the rocks are not considered boundaries in the ground truth. Moreover, the edges are as strong around the penguin as they are around the rocks, which are less prominent in human vision.

Fig. 22. Canny edges of Image 7

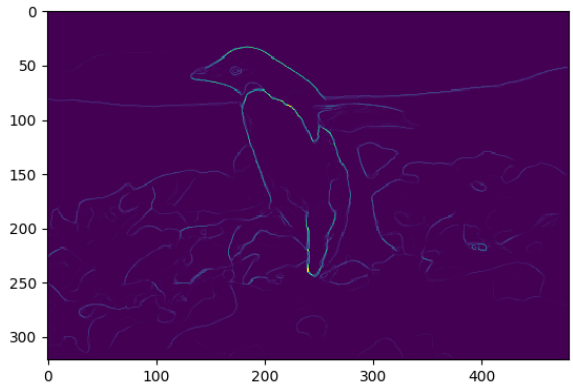

Fig. 23. Pb-lite output

M. Pb-lite Output

We combine all the signals described earlier to create a holistic prediction of probability of boundary.

1) Interpretation and Analysis: Notice that now we have solved several of these problems. The Pb-lite output now is very high around the edges of the penguin, and attains the highest value in the image around the penguin. Pb-lite is also able to detect the boundary in the background, unlike the Sobel baseline. Boundaries around rocks are also less prominent, as one would naturally expect. An additional positive sign is the continuity of the edges around the penguin, which capture the closed physical body of the penguin better than the Canny baseline.

By incorporating brightness, color, and texture gradient information with hundreds of filters at varying scale and orientation, the Pb-lite is able to capture a wide range of information that is left out by any single component, as evident in the Sobel and Canny baseline. By expressing the probability of boundary as a linear combination with weights summing to 1, the computation of the probability by Pb-lite effectively considers weighted votes by each of the texton gradient, brightness gradient, color gradient, CannyPb, and SobelPb (the latter two with variable weights), each of which is able to detect boundaries of a particular type, but none of which is able to detect boundaries of all types.

II. Phase 2: Deep Dive on Deep Learning

A. Section 3.3

Number of parameters = 3247422

Optimizer = Adam Optimizer Learning rate = 1e-3 Batch size = 16

B. Section 3.4

Number of parameters = 3250806

Optimizer = Adam Optimizer Decaying learning rate Initially 1e-3 Drops by factor or 1e-1 after 20 epochs Drops by factor of 1e-2 after 40 epochs Drops by factor of 1e-3 after 60 epochs Drops by factor of 0.5e-3 after 80 epochs Batch size = 32

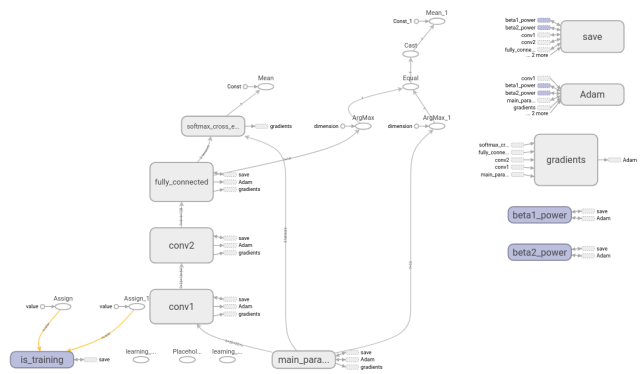

Fig. 24. CNN Architecture

C. Experiments

In this section I experimented with different learning rates. The learning rate is variable. I also used batch normalization between layers. This led to an increase in accuracy.

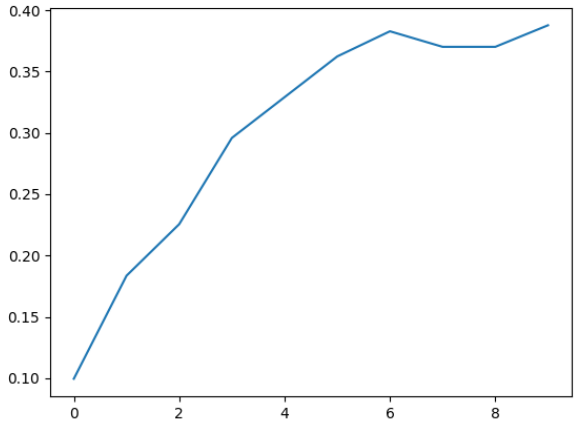

Fig. 25. CNN Train accuracy over epochs

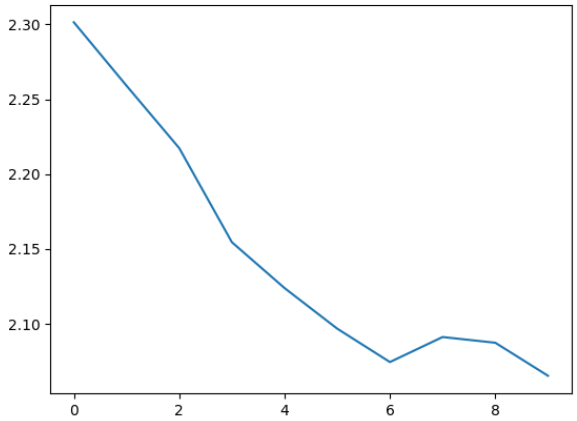

Fig. 26. CNN Train loss over epochs

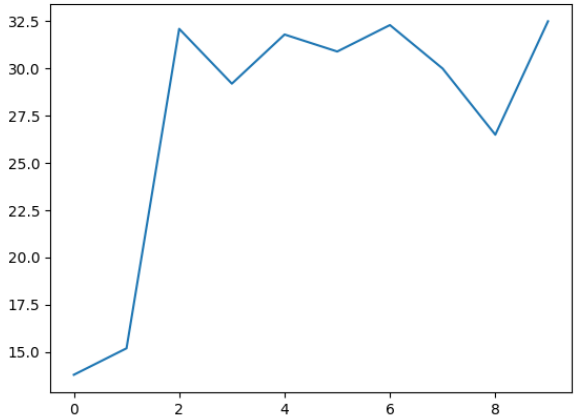

Fig. 27. CNN Test accuracy over epochs

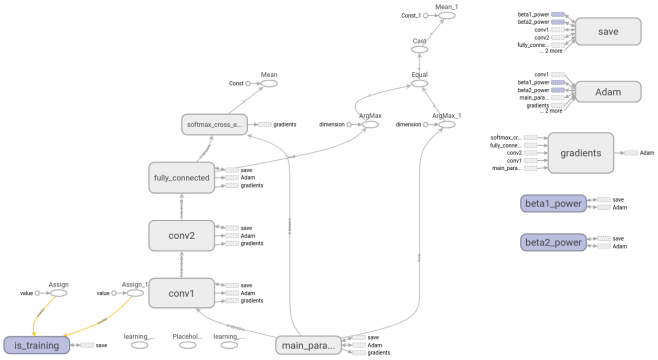

Fig. 28. CNN Architecture

64	5	3	4	2	7	5	5	1	7
6	38	3	4	0	11	0	2	2	23
17	5	34	4	14	13	9	4	0	0
13	5	28	12	6	15	19	5	0	0
9	3	19	3	26	6	12	9	0	3
3	1	25	14	5	18	7	11	0	2
4	3	24	7	24	7	38	2	0	3
15	3	19	4	8	8	3	41	0	1
49	12	10	7	0	4	0	4	4	16
16	24	2	5	0	3	4	3	2	50

Fig. 29. CNN Confusion matrix