CV HW 0: Alohomora

Jesdin Raphael Computer Science
Worcester Polytechnic Institute
Email: jraphael@wpi.edu

***Abstract*—In this assignment, I learned about different types of filters for finding the Boundary. I used the Oriented Derivative of Gaussian (DoG) filters, Leung-Malik (LM) filters and the Gabor filters to implement the pb_lite filter.**

I. Phase1

For creating pb_lite boundary output the first step of the pb lite boundary detection pipeline is to filter the image with a set of filter banks[1]. First three different sets of filter banks were created: Oriented Derivative of Gaussian filter bank, Leung-Malik filter bank, and Gabor Filter Bank. Using these filter banks and half disk masks I generated a Texture Map, Brightness Map and Color Map which were segregated into bins using Chi Square distance.

A. Filter Banks:

Created three sets of filter banks. Each set helps capture texture and orientation information in the image.

1) Oriented Derivative of Gaussian (DoG) filters:
   - These filters capture edge information at various orientations and scales.
   - It can be calculated by convolving the sobel filter with the gaussian kernel and then rotating it at some angle 'o'.
   - Fig 1. shows the implemented DoG filters with 2 scales and 16 orientations.

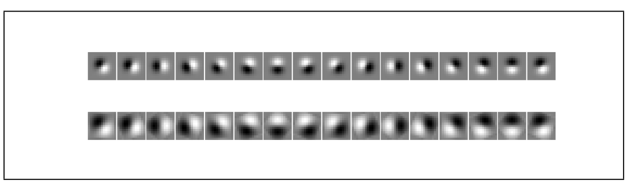

Fig. 1: Oriented DoG Filters.

2) Leung-Malik (LM) filters:
   - These filters provide a comprehensive set of multi-scale, multi-orientation filters.
   - The LM filter bank comprises a variety of filters, including edge, bar, and spot filters, distributed across various scales and orientations.
   - It encompasses a total of 48 filters, consisting of 2 Gaussian derivative filters with 6 orientations and 3 scales, along with 8 Laplacian of Gaussian filters and 4 Gaussian filters [2].
   - Two versions of LM filters were implemented. LMs (LM Small) and LML (LM Large) which used

     $$\sigma = [1, \sqrt{2}, 2, 2\sqrt{2}]$$

     and

     $$\sigma = [\sqrt{2}, 2, 2\sqrt{2}, 4]$$

     respectively.
   - Fig 2. shows the 48 implemented filters of the Leung-Malik filter bank.

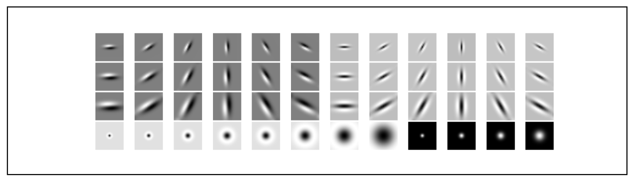

Fig. 2: Leung-Malik Filters.

3) Gabor filters:
   - These filters are inspired by the human visual system, and are used for texture analysis.
   - Eqn 1. shows the formula for the Gabor Filter [3].
   - Fig 3. shows the 64 implemented Gabor Filters.

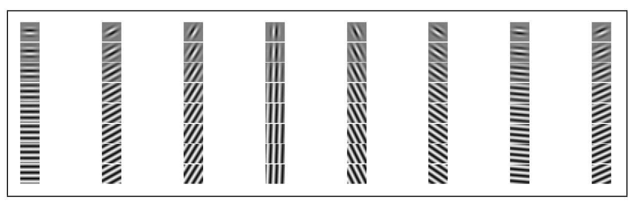

Fig. 3: Gabor Filters.

$$G(x, y) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda} + \psi\right) \qquad (1)$$

$Where : \lambda$: Wavelength
θ : Orientation
ψ : Phase offset
σ : Standard deviation of the Gaussian
γ : Aspect ratio

B. Texton Map (T)

By filtering the image with the above filter banks, we get a set of filter responses. These responses are then clustered using k-means to create a texton map, which encodes the texture information in the image. I have used K=64 clusters.

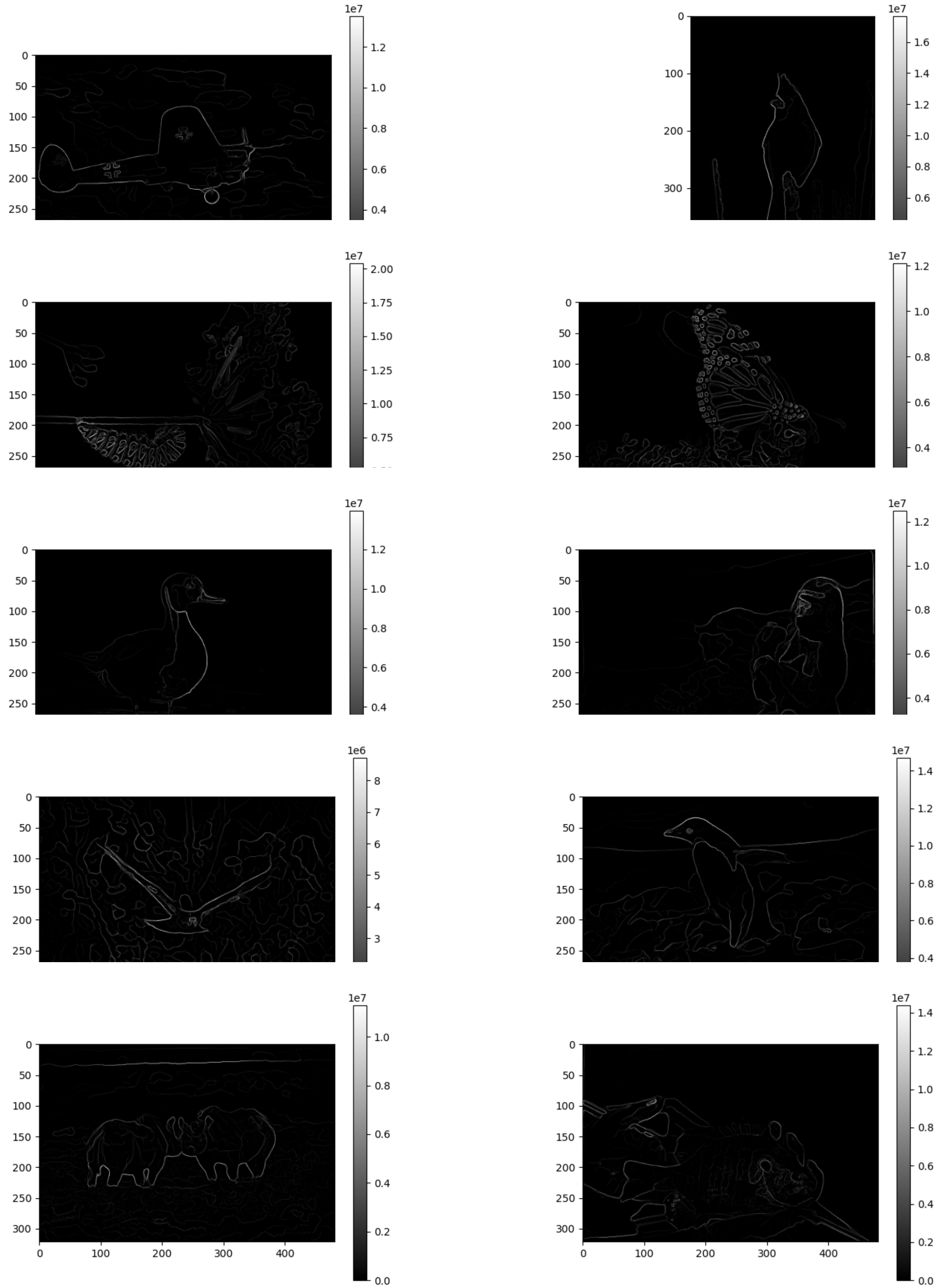

Fig. 4: Output of PB Lite.

C. Brightness Map (B)

Similar to Texton Map, here we cluster the Brightness values in the image. I have used K=16 clusters.

D. Color Map (C)

Similar to Brightness Map, here we cluster the Color values in the image. I have used K=16 clusters.

E. Texture, Brightness, and Color Gradients (T_g, B_g, C_g)

The gradients for Texture, Brightness, and Color are calculated by using the χ^2 formula as shown in the equation 2 [1]. where K is the number of clusters or bins. g_i and h_i are two half-disk masks. This is done for every filter and then the average is taken to obtain the gradient.

$$\chi^2(g,h) = \frac{1}{2}\sum_{i=1}^{K}\frac{(g_i - h_i)^2}{g_i + h_i} \tag{2}$$

F. Pb-lite Output

Once the gradients have been obtained we can get the Pb-lite output with the gradients and the Canny and Sobel Baselines using the Eqn. 3 [1]. The weights used in this equation was $w1 = w2 = 0.5$. The output of the PB Lite method is shown in Fig 4. The images are ordered from left to right and top to bottom.

$$PbEdges = (Tg+Bg+Cg)^3 \odot (w1 \cdot \text{cannyPb} + w2 \cdot \text{sobelPb}) \tag{3}$$

G. ***Conclusion***

I believe that the output of the Pb-lite is better than the Canny and Sobel Baselines because:

1) In the canny baseline Though the main boundaries are clear there is a lot of noise (unnecessary boundaries) in the image.
2) In the Sobel baseline though there is no noise in most cases more than half the boundaries are not visible
3) Pb lite combines the advantages of both Canny and Sobel by giving a complete border for the object (light in some places) but reduces noise.
4) The weights used were 0.5 for Canny and 0.5 for Sobel.
5) increasing the weight for the Canny Baseline would result in a more thicker or clearer output but would also increase the noise.

II. Phase2

A. ***3.3. Train your first neural network***

1) Plot of Train Accuracy over Epochs (Fig 5)
2) Plot of Test Accuracy over Epochs (Fig 6)
3) Number of Parameters in Model: 8
4) Plot of loss over Epochs (Fig 7)
5) An Image of the architecture (Fig 8)
6) Optimizer chosen: Adam Optimizer. Learning Rate = $1e-5$
7) Batch size chosen: 64

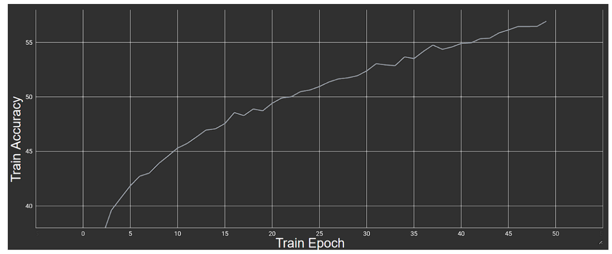

Fig. 5: Train Accuracy over Epoch.

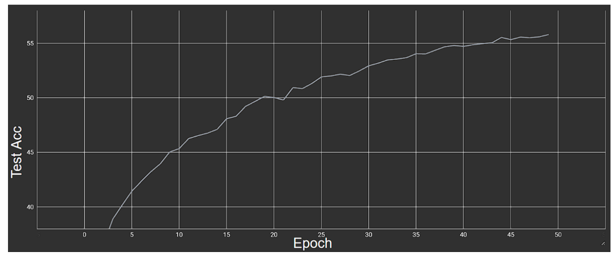

Fig. 6: Test Accuracy over Epoch.

8) Number of Epochs: 50
9) Confusion Matrix of the trained model on training data (Table I)
   Accuracy: 56.812
10) Confusion Matrix of the trained model on testing data (Table III)
   Accuracy: 55.949

B. 3.4 Improving Accuracy of your neural network

- Plot of Train_Accuracy over Epochs (Fig 9
- Plot of Test_Accuracy over Epochs (Fig 10)
- Number of Parameters in your model: 20
- Plot of loss over Epochs (Fig 11)
- An Image of the architecture (Fig 12)
- Optimizer chosen: Adam. Learning Rate = $1e-5$
- Batch size chosen: 64 (first half of epochs) / 128 (second half of epochs)
- Confusion Matrix of the trained model on training data Accuracy: 47.17%
- Confusion Matrix of the trained model on testing data (Table IV) Accuracy: 46.584
- A detailed analysis of all the tricks used.
  To improve Accuracy I have done the following
  1) Normalize Data.

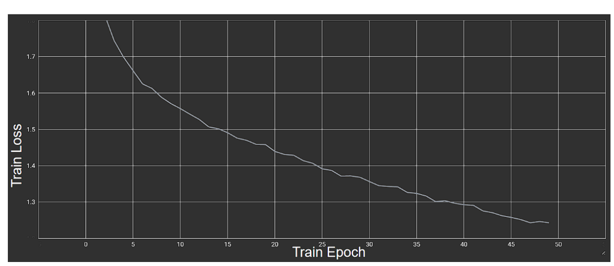

Fig. 7: Train Loss over Epoch.

Actual	Predicted (0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
(0)	3121	184	271	121	102	84	85	123	609	298
(1)	211	3316	24	61	27	36	122	107	251	841
(2)	420	63	1913	345	651	510	528	355	104	108
(3)	102	52	326	1942	336	1118	579	305	81	159
(4)	213	37	492	306	2282	346	589	566	69	98
(5)	40	42	346	866	268	2568	261	473	45	90
(6)	40	55	275	328	419	163	3426	141	49	103
(7)	95	44	170	281	413	415	129	3200	44	208
(8)	576	283	77	86	60	67	55	38	3414	343
(9)	244	720	45	111	37	66	165	192	204	3215

TABLE I: Confusion Matrix for Train

Actual	Predicted (0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
(0)	621	33	62	27	18	19	30	16	121	51
(1)	46	640	9	11	4	6	19	25	60	179
(2)	75	12	368	72	134	109	106	81	24	18
(3)	24	10	64	378	69	222	118	60	13	39
(4)	34	8	111	71	432	81	124	109	18	12
(5)	11	4	76	162	55	511	51	101	16	10
(6)	8	7	52	65	70	40	701	25	8	24
(7)	19	3	32	60	75	85	35	633	12	43
(8)	112	54	13	19	12	10	12	17	676	72
(9)	52	135	11	18	9	14	34	48	53	626

TABLE II: Confusion Matrix for Test

Actual	Predicted (0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
(0)	1336	144	635	738	288	18	230	143	1148	318
(1)	77	2597	147	262	170	10	394	37	392	910
(2)	90	36	2283	1141	655	230	221	114	181	46
(3)	11	20	312	3396	359	307	375	75	107	38
(4)	50	9	1152	949	2164	125	253	166	89	41
(5)	5	14	526	2122	451	1439	260	115	51	16
(6)	7	30	509	1224	695	57	2350	12	93	22
(7)	22	21	397	886	962	369	259	1960	50	73
(8)	138	154	183	389	94	17	96	49	3468	411
(9)	98	610	173	412	180	37	513	129	263	2584

TABLE III: Confusion Matrix for Test

Actual	Predicted (0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
(0)	275	19	124	149	51	2	60	25	232	61
(1)	22	508	36	49	26	3	76	6	86	187
(2)	28	5	423	240	118	53	58	25	39	10
(3)	2	7	76	644	83	72	64	15	24	10
(4)	14	4	221	216	426	31	44	21	20	3
(5)	2	3	104	418	98	284	42	22	16	8
(6)	2	4	93	239	142	14	488	1	13	4
(7)	3	4	73	178	189	76	53	402	5	14
(8)	30	35	35	79	24	4	13	11	693	73
(9)	19	125	21	94	31	3	95	33	71	508

TABLE IV: Confusion Matrix for Test on Improved Model

2) Augment Data by adding a RandomHorizontalFlip and RandomCrop.
3) Trained by updating Minibatch Size. First 25 epochs had minibatch size as 64 while the next 25 epochs had minibatch size as 128.
4) Added Batch Normalization after each Convolution Layer.
5) Added 2 Fully Connected Layers before Output layer.

C. *3.5.1 ResNet*

- Plot of Train_Accuracy over Epochs (Fig 13)
- Plot of Test_Accuracy over Epochs (Fig 14)
- Number of Parameters in your model: 122
- Plot of loss over Epochs (Fig 15)
- An Image of the architecture (Fig 16)
- Optimizer chosen: Adam with Learning Rate = $1e-5$
- Batch size chosen: 64
- Confusion Matrix of the trained model on training data (Table V Accuracy: 52.79%
- Confusion Matrix of the trained model on testing data

Actual \ Predicted	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
(0)	2613	195	227	129	337	281	114	52	782	268
(1)	384	3488	92	57	61	44	103	47	239	481
(2)	281	109	1557	627	1102	524	300	234	140	123
(3)	116	81	349	2192	416	995	421	274	78	78
(4)	141	79	422	495	2717	250	291	430	83	90
(5)	52	64	348	1044	490	2290	166	446	38	61
(6)	30	83	340	654	795	145	2725	108	39	80
(7)	95	67	219	287	698	429	98	2948	28	130
(8)	779	265	96	119	157	140	85	21	2815	522
(9)	311	797	106	117	88	149	106	122	161	3042

TABLE V: Confusion Matrix for Train

(Table VI Accuracy: 47.05%

D. 3.5.2 ResNeXt

- Plot of Train_Accuracy over Epochs (Fig 17)
- Plot of Test_Accuracy over Epochs (Fig 18)
- Number of Parameters in your model: 320
- Plot of loss over Epochs (Fig 19)
- Plot of Test_Accuracy over Epochs (Fig 19)
- An Image of the architecture (Fig 20) [4]. As Netron produced too big an image I had to get the architecture from the paper.
- Optimizer chosen: Adam Optimizer with Learning Rate $1e-5$
- Batch size chosen: 64
- Confusion Matrix of the trained model on training data (Table VII)
- Confusion Matrix of the trained model on testing data (Table VIII Test Accuracy: 21.95%

E. 3.5.3 DenseNet

I have been unable to debug the error I got while Training the DenseNet. It has Something to do with the shape I Believe.

F. Conclusion

From Table IX we can see that the performance increases till ResNet and for ResNeXt it starts decreasing. This can be that after ResNet the model has become overcomplicated. Also the train time has increased in proportion to the number of parameters. Thus it would be better to choose between the Improved Base Model and ResNet after tuning the model and training it till there is no improvement rather than a limited number of epochs.

Acknowledgment

The authors would like to thank...

References

[1] R. . R. Perception and M. Learning, "Rbe 549 spring 2024 - homework 0," https://rbe549.github.io/spring2024/hw/hw0/#report, accessed: Jan 2024.
[2] U. o. O. Visual Geometry Group. (Year of Access) Texture classification using convolutional neural networks. [Online]. Available: https://www.robots.ox.ac.uk/~vgg/research/texclass/filters.html
[3] W. contributors, "Gabor filter," https://wikipedia.org/wiki/Gabor_filter, accessed: Jan 2024.
[4] S. Zagoruyko and N. Komodakis, "Aggregated residual transformations for deep neural networks," *arXiv preprint arXiv:1611.05431*, 2016.

	Predicted									
Actual	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
(0)	481	37	61	31	62	66	22	6	178	54
(1)	92	606	25	11	13	7	34	15	54	142
(2)	66	17	268	127	230	109	64	59	28	31
(3)	21	26	87	362	104	210	87	60	21	19
(4)	30	15	89	110	484	57	78	105	12	20
(5)	15	5	62	223	95	420	32	120	11	14
(6)	6	14	74	148	151	29	516	21	11	30
(7)	22	12	63	76	140	107	20	509	8	40
(8)	172	67	23	23	33	27	22	4	516	110
(9)	79	179	18	34	18	28	22	34	53	535

TABLE VI: Confusion Matrix for Test on Resnet Model

	Predicted									
Actual	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
(0)	1201	493	383	269	163	160	121	565	780	863
(1)	383	1472	240	264	228	256	415	503	200	1035
(2)	272	173	1074	910	529	608	394	479	172	386
(3)	158	256	568	1044	429	824	640	581	74	426
(4)	184	155	964	1093	710	640	486	391	91	284
(5)	135	226	665	863	447	1064	640	629	50	280
(6)	91	168	711	1180	624	601	995	332	30	267
(7)	218	266	536	615	508	505	419	1470	70	392
(8)	715	795	248	261	127	178	70	379	979	1247
(9)	361	988	257	210	201	169	246	874	172	1521

TABLE VII: Train Confusion Matrix for ResNeXt Model

	Predicted									
Actual	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
(0)	210	100	86	53	22	48	25	120	176	158
(1)	80	261	62	70	50	59	71	114	43	189
(2)	61	34	210	177	102	117	84	99	38	77
(3)	42	61	111	197	80	163	125	127	13	78
(4)	33	27	195	235	128	136	96	86	11	53
(5)	22	42	119	185	89	219	98	155	19	49
(6)	21	32	152	213	130	141	198	49	5	59
(7)	38	53	90	125	99	102	85	308	10	87
(8)	139	144	53	56	35	41	22	75	175	257
(9)	63	224	37	51	45	35	50	171	38	286

TABLE VIII: Test Confusion Matrix for ResNeXt Model

Model	Number of Parameters	Final Train Accuracy	Final Test Accuracy	Inference Run-Time (min)	Epochs
Base	8	56.812	55.959	13	50
Improved Base	20	47.17	46.584	20	50
ResNet	122	52.79	47.05	120	20
ResNeXt	320	23.067	21.95	195	20

TABLE IX: Model Comparison

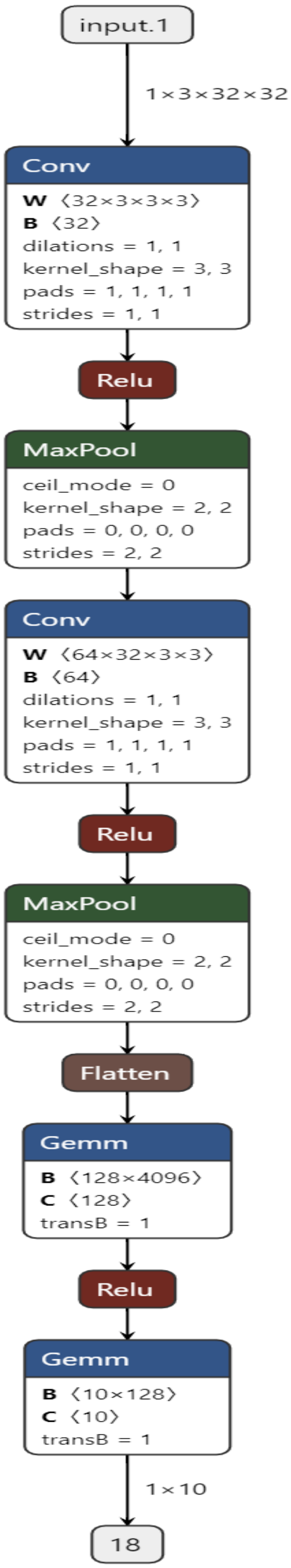

Fig. 8: Model Architecture.

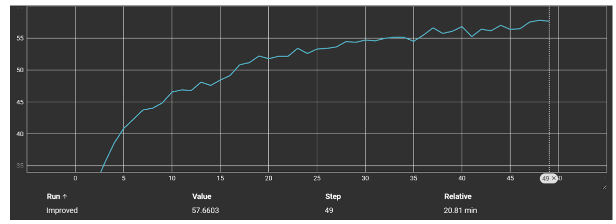

Fig. 9: Train Accuracy over Epoch for Improved model.

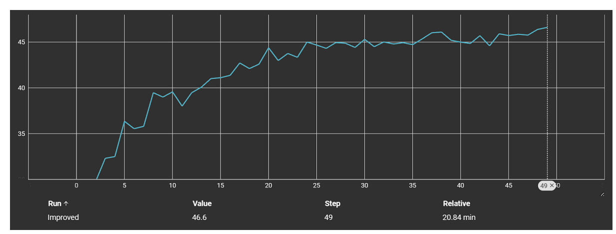

Fig. 10: Test Accuracy over Epoch for Improved model.

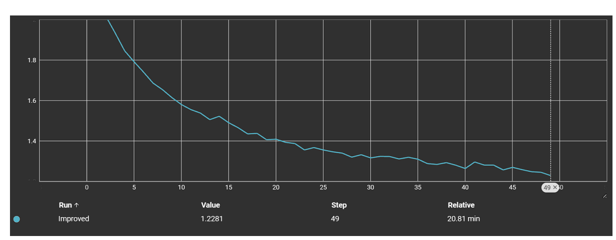

Fig. 11: Train Loss over Epoch for Improved model.

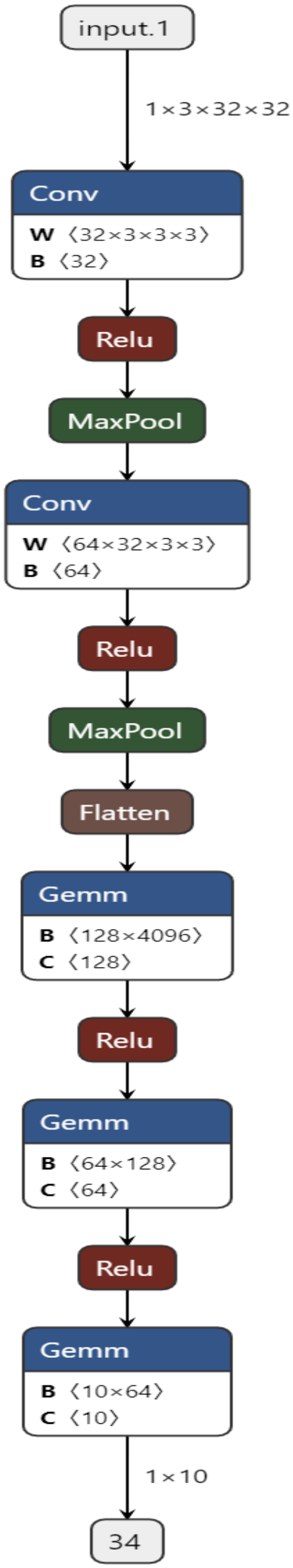

Fig. 12: Improved Model Architecture.

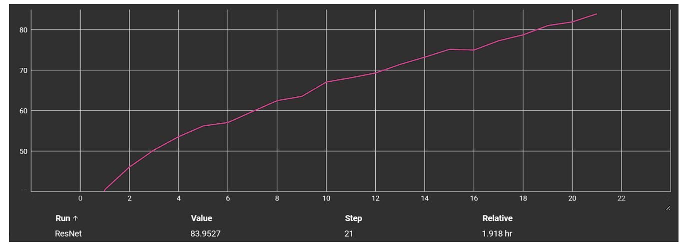

Fig. 13: Resnet Train Acc.

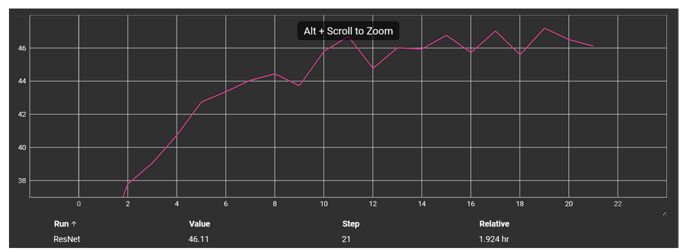

Fig. 14: Resnet Test Acc.

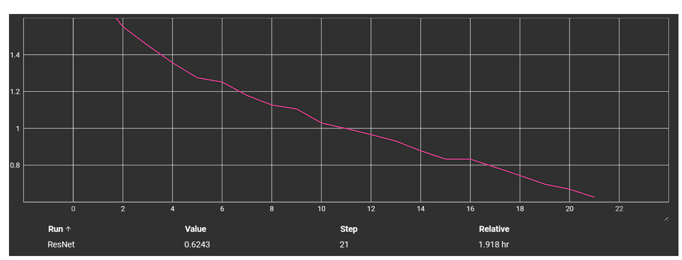

Fig. 15: Resnet Loss.

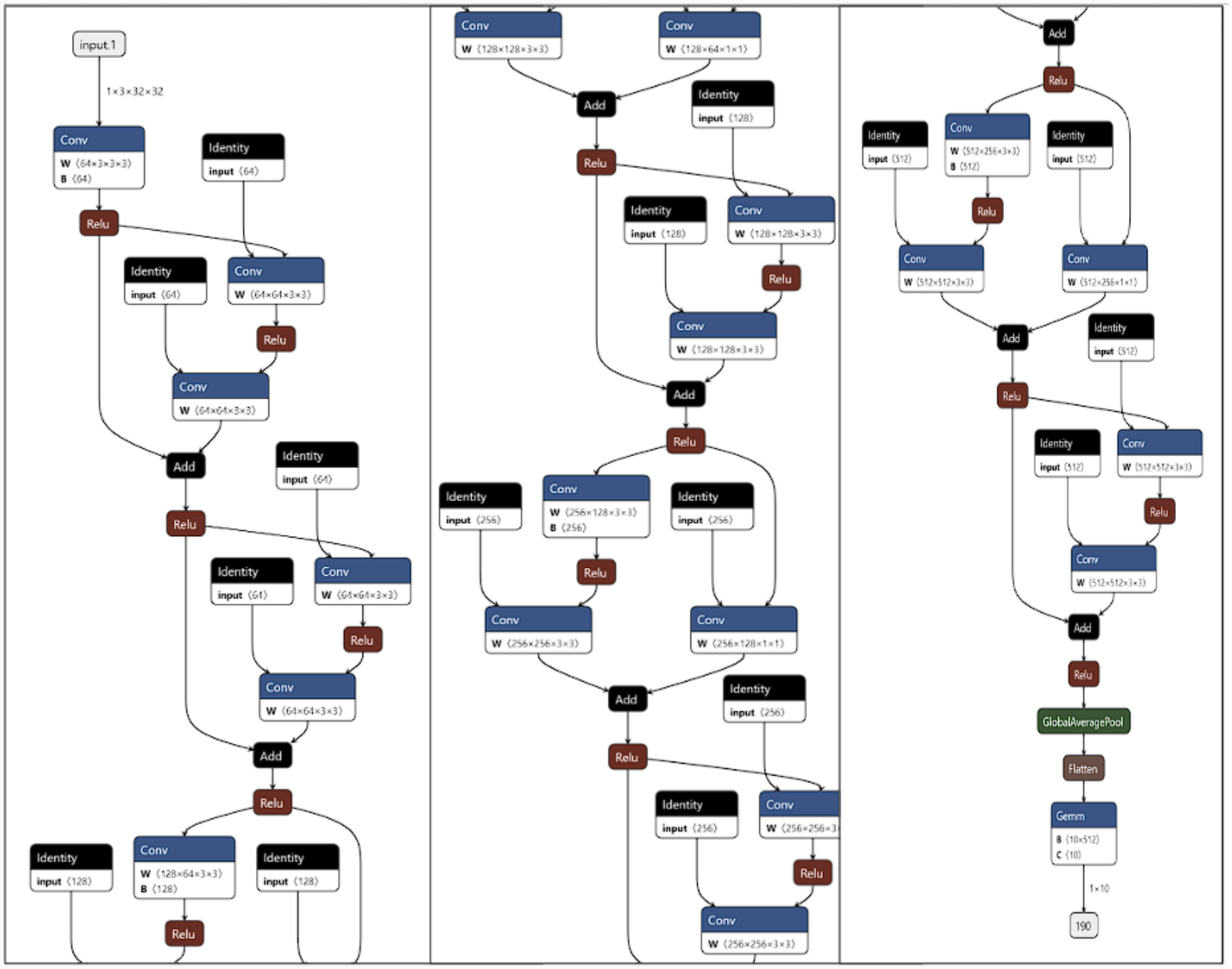

Fig. 16: Resnet Architecture.

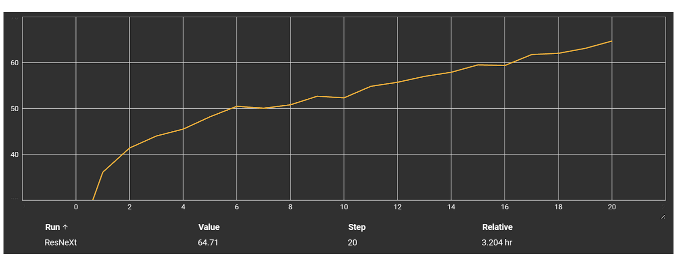

Fig. 17: ResNeXt Train Accuracy Over Epoch

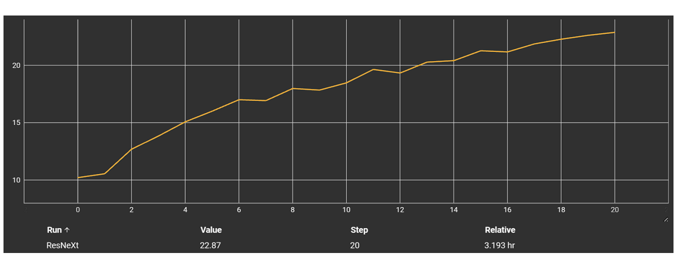

Fig. 18: ResNeXt Test Accuracy Over Epoch

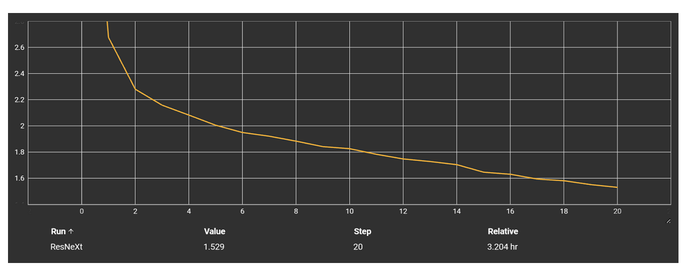

Fig. 19: ResNeXt Loss Over Epoch

stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C{=}32 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C{=}32 \\ 1\times1, 512 \end{bmatrix} \times 4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C{=}32 \\ 1\times1, 1024 \end{bmatrix} \times 6$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C{=}32 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		$\mathbf{25.5}\times10^6$	$\mathbf{25.0}\times10^6$
FLOPs		$\mathbf{4.1}\times10^9$	$\mathbf{4.2}\times10^9$

Fig. 20: ResNeXt Loss Over Epoch