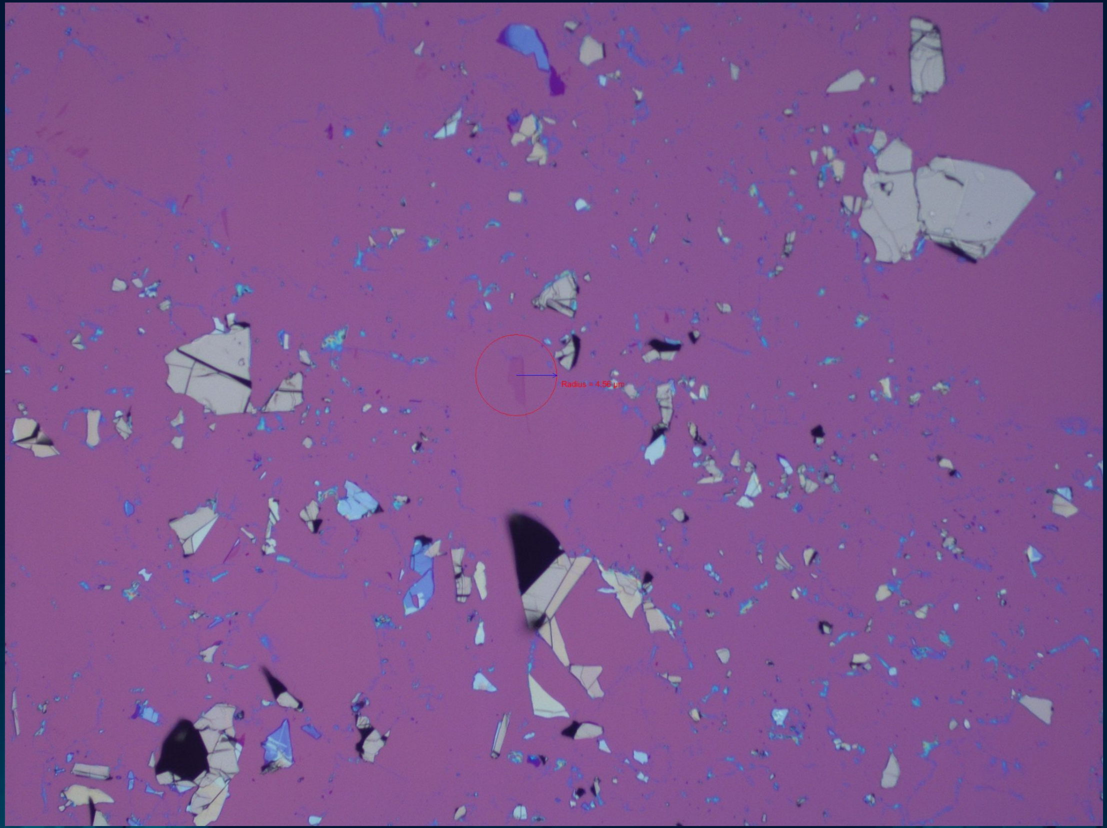
Binary Classification of Graphene Flakes

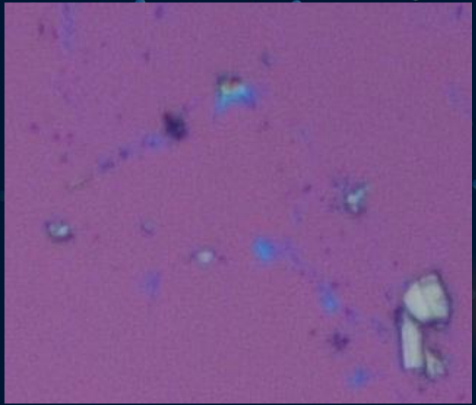
Samuel Rager and Alvin Wang

Main References

- VII. Krizhevsky, Sutskever, & Hinton. “ImageNet classification with deep convolutional neural networks.” *Commun. ACM 60*. 2017, accessed 29 October 2024, <https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.
- VIII. Kingma, Diederik P. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014), accessed 29 October 2024, <https://arxiv.org/pdf/1412.6980>.
- XI. He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, accessed 29 October 2024, <https://arxiv.org/pdf/1512.03385v1>.

Data Cleaning Methodology

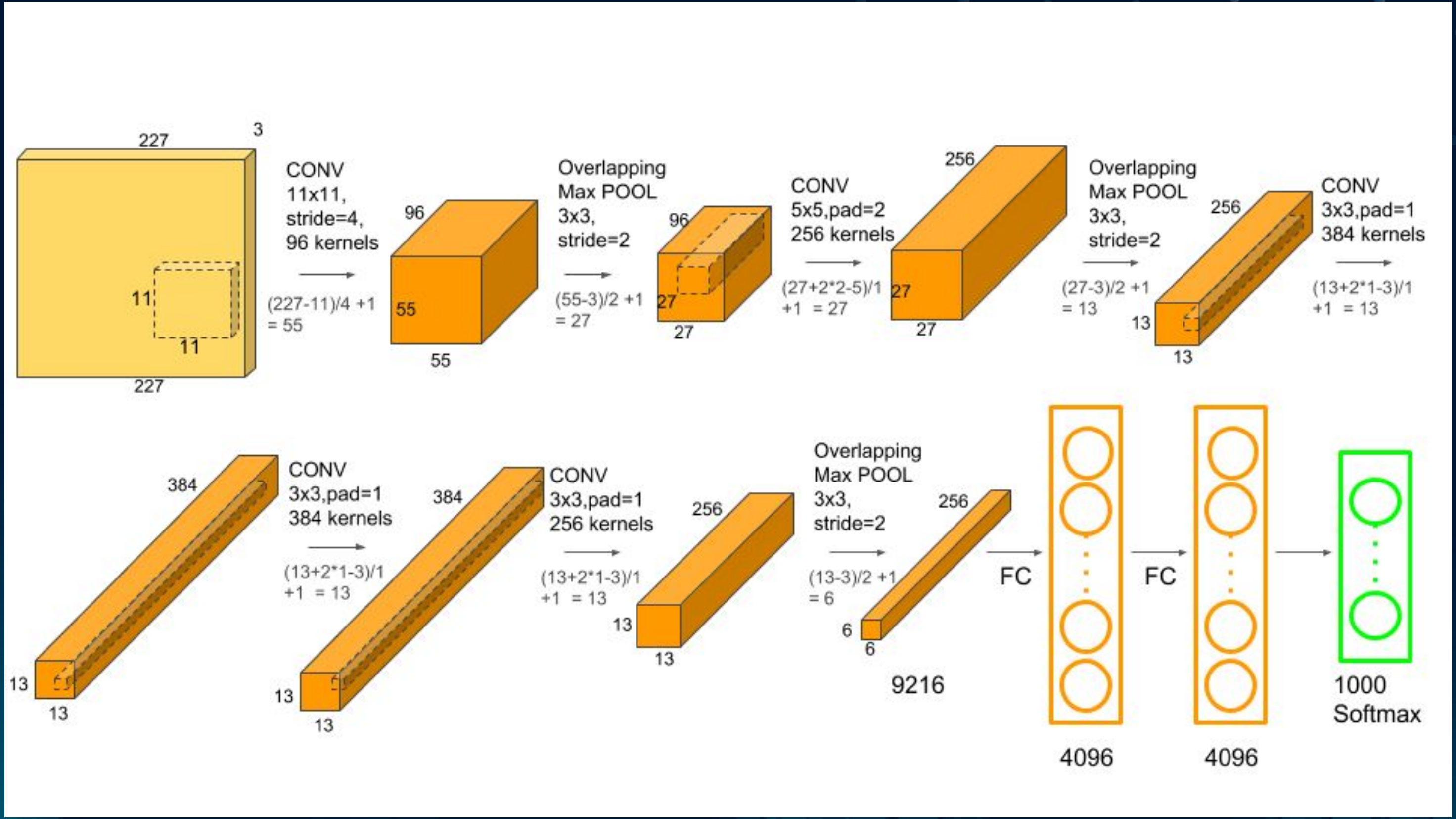




Timeline

2010	2012	2014	2015	2016	2017	2018	2019	2023
• First ImageNet challenge	• AlexNet	• Adam • VGG • GoogleNet	• ResNet • Batch Norm. • GoogleNet renamed Inception • Yolo V1	• Inception v3	• AdamW • Transformer • Last Imagenet Challenge	• Mobile Net v2 (Inverted Residual)	• Efficient Net	• Yolo V8

AlexNet



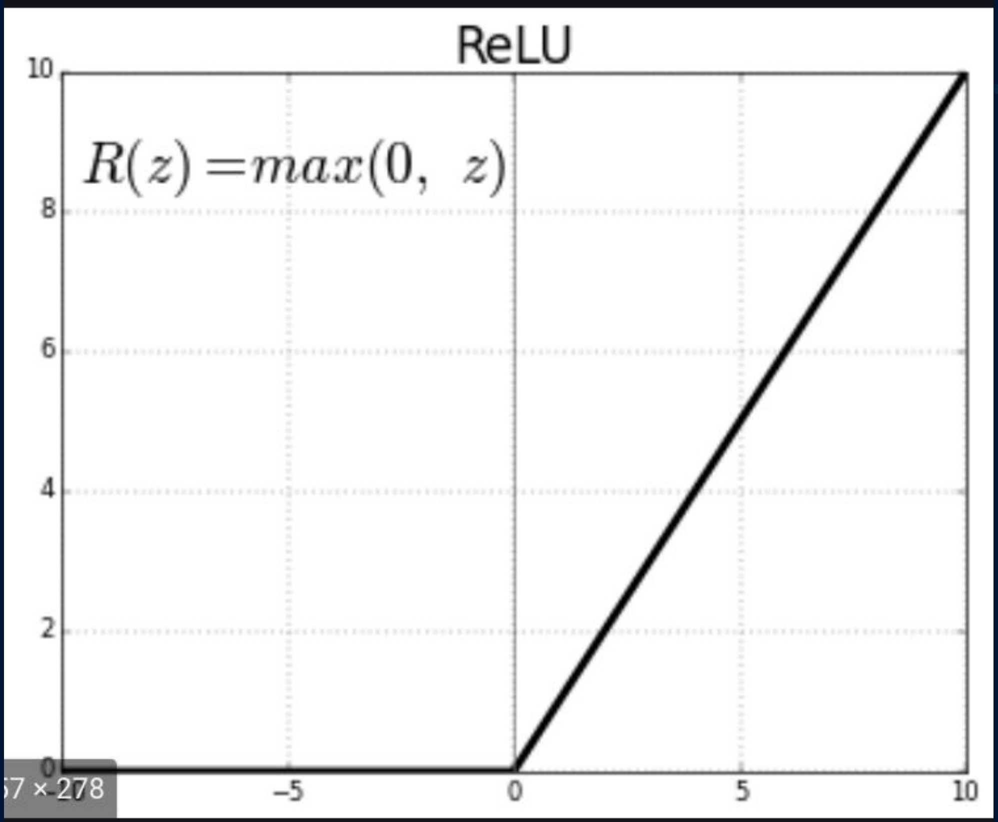
227
3
227
11
11
CONV
11x11,
stride=4,
96 kernels
(227-11)/4 +1
= 55
96
55
55
Overlapping
Max POOL
3x3,
stride=2
(55-3)/2 +1
= 27
96
27
27
CONV
5x5,pad=2
256 kernels
(27+2*2-5)/1
+1 = 27
256
27
27
Overlapping
Max POOL
3x3,
stride=2
(27-3)/2 +1
= 13
256
13
13
CONV
3x3,pad=1
384 kernels
(13+2*1-3)/1
+1 = 13
384
13
13
CONV
3x3,pad=1
384 kernels
(13+2*1-3)/1
+1 = 13
384
13
13
CONV
3x3,pad=1
256 kernels
(13+2*1-3)/1
+1 = 13
256
13
13
Overlapping
Max POOL
3x3,
stride=2
(13-3)/2 +1
= 6
256
6
6
9216
FC
4096
FC
4096
1000
Softmax

I.

Convolution

Max Pooling

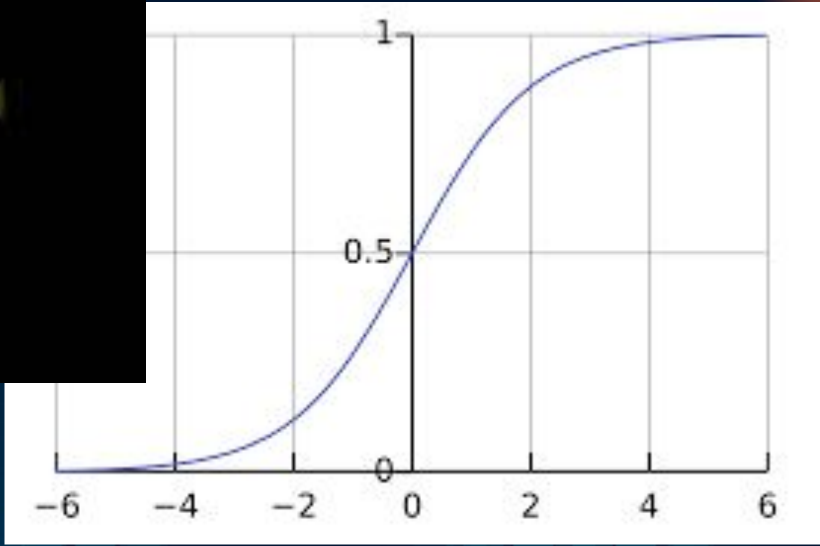
Rectified Linear Function



IV.

Softmax- Reduces to Sigmoid in Binary Case

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



V. & VI.

Local Response Normalization

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0, i-n/2)}^{\min(N-1, i+n/2)} (a^j_{x,y})^2 \right)^\beta$$

For AlexNet, k=2, n=5, alpha=10^-4, beta=0.75

Data Structure for Colab Models

Data Structure

```
Dataset-|
          | _>train---| _> graphene
                      |_> non_graphene

          | _>test---| _> graphene
                     |_> non_graphene
```

Adam (Adaptive Moment Estimation/ Gradient Descent with Momentum)

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Backpropagation

Algorithm 1 Backpropagation

procedure BACKPROPAGATION(X, θ)

  $a^0 \leftarrow x$ ▷ Set Input

  for $l \leftarrow [1, L]$ **do** ▷ Forward Pass: Compute weighted inputs and activations

    $z^l \leftarrow W^l a^{l-1} + b^l$

    $a^l \leftarrow \sigma(z^l)$

  end for

  $\delta_L \leftarrow (a^L - y) \odot \sigma'(z^L)$ ▷ Compute Error in Final Layer

  for $l \leftarrow [L-1, 1]$ **do** ▷ Backpropagate Error

    $\delta^l \leftarrow (W^{l+1})^T \delta^{l+1} \odot \sigma'(z^l)$

  end for

  for $l \leftarrow [1, L]$ **do** ▷ Compute gradients w.r.t. network parameters

    $\nabla_{b^l} = \delta^l$

    $\nabla_{W^l} = \delta^l (a^{l-1})^T$

  end for

  return $\nabla_\theta = (\nabla_{W^l}, \nabla_{b^l})$

end procedure

IX.

Binary Cross Entropy Loss

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

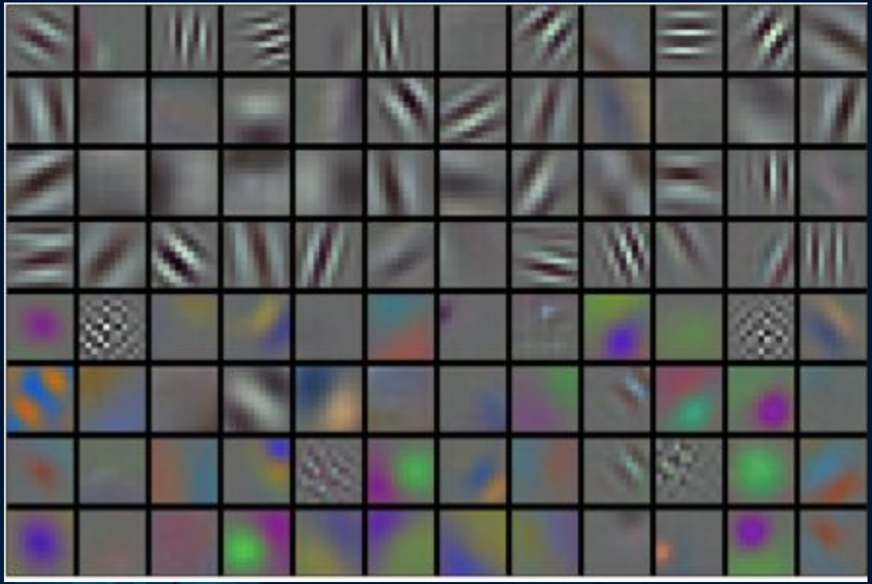
> N *is the number of samples*
> y_i i *is the true label for the* i^{th} *sample* (0 *or* 1)
> p_i *is the predicted probability that the* i^{th} *sample belongs to Positive class*

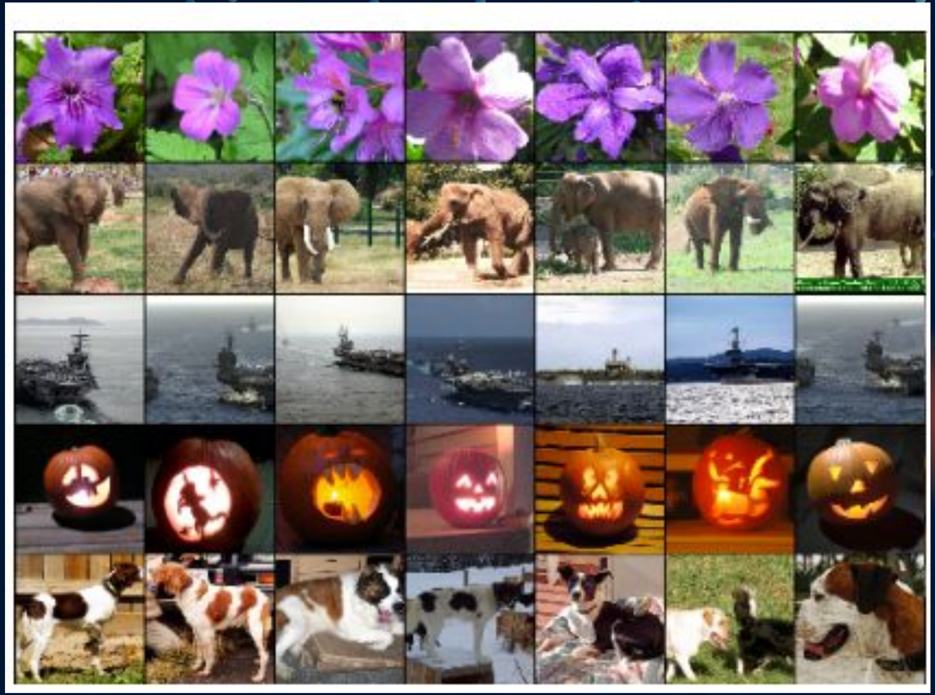
Binary Cross Entropy with Logits Loss

$$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N}\left[\max(z_i, 0) - z_i y_i + \log\left(1 + e^{-|z_i|}\right)\right]$$

AlexNet

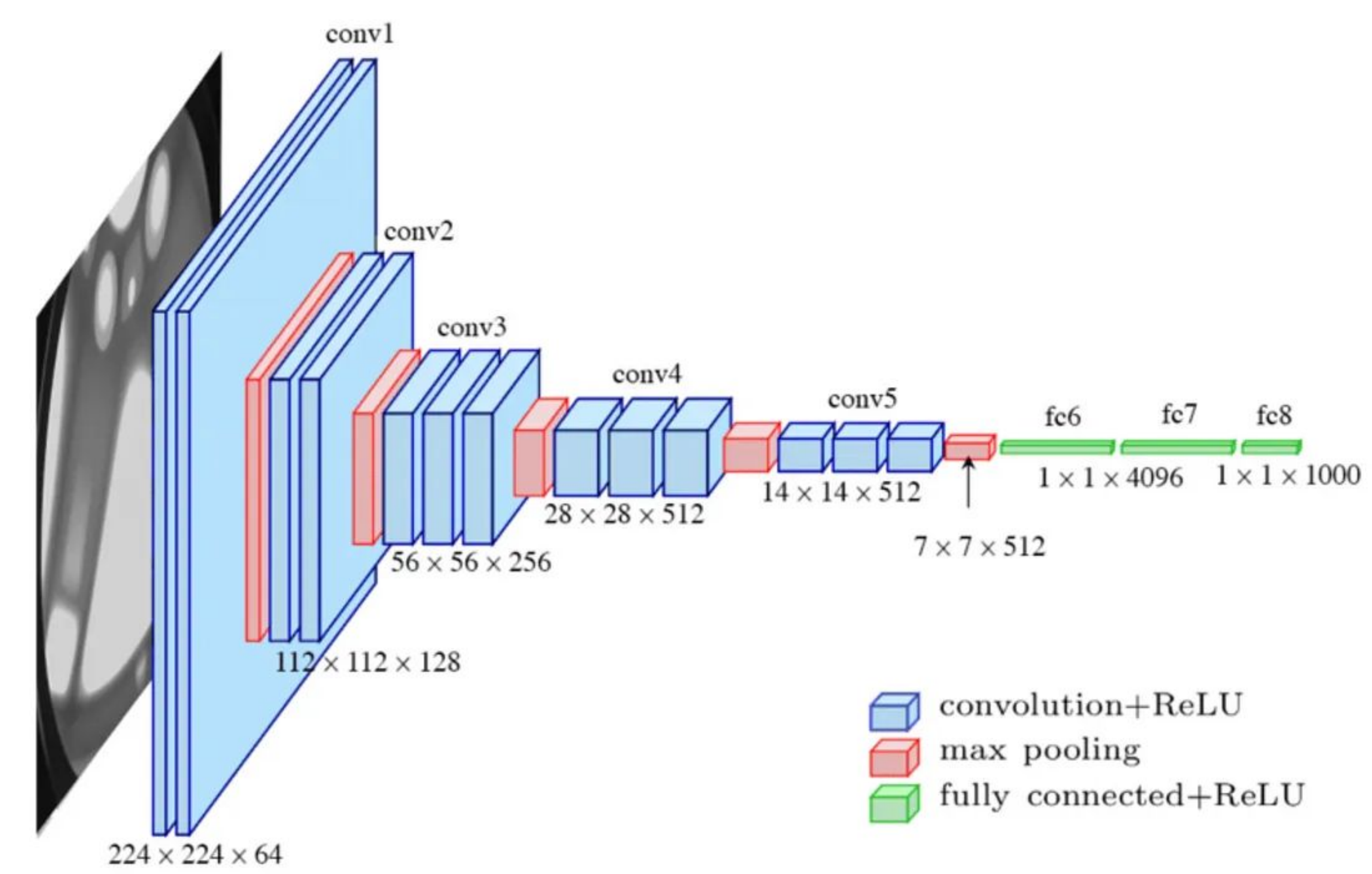
Contribution Mnemonic: Programming working backwards from desired output





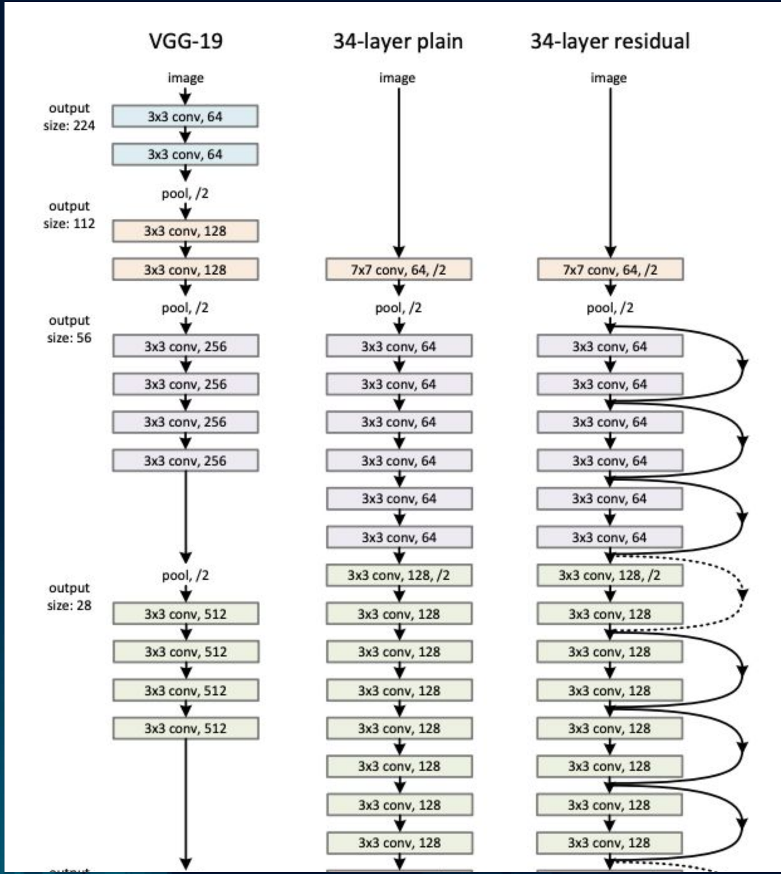
VII.

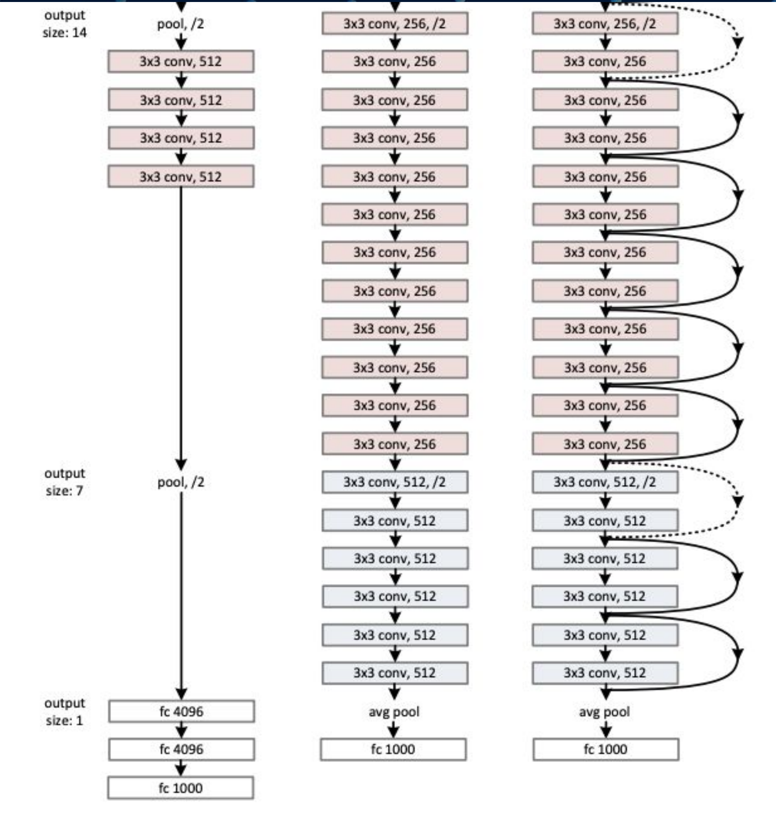
Visual Geometry Group (VGG) 16



X.

ResNet Architecture





ResNet 50 Architecture

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$

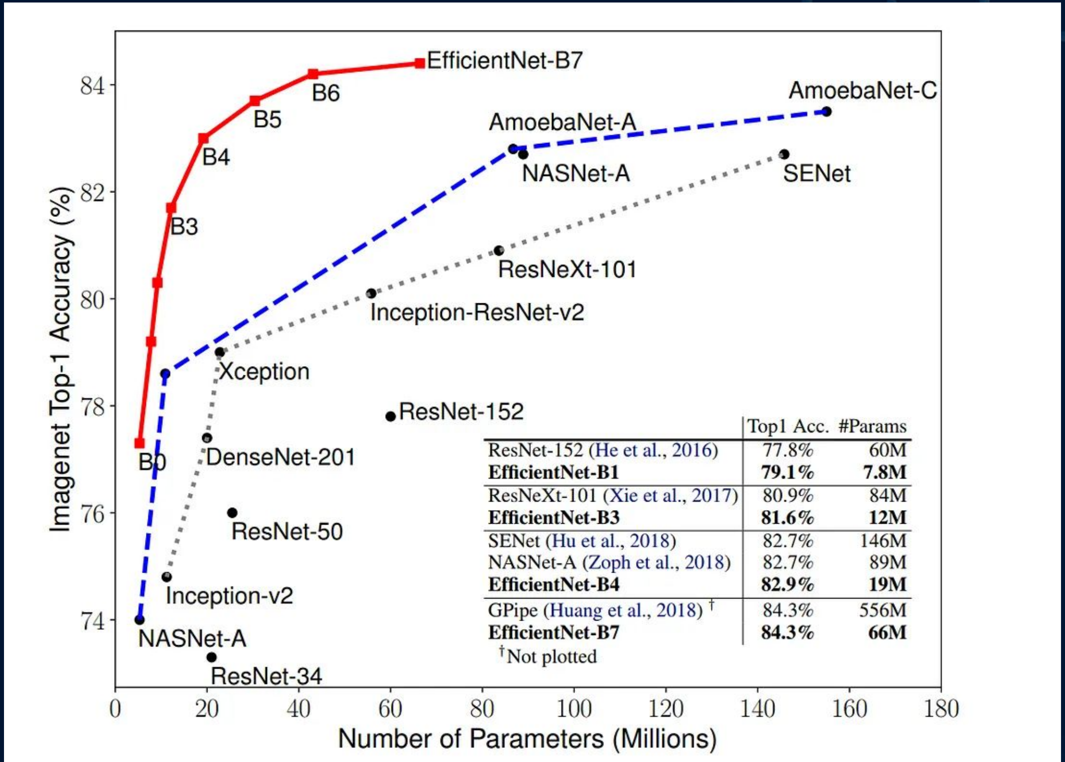
$N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$ // Training BN network
for $k = 1 \ldots K$ **do**
    Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)
    Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
end for
Train $N_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
$N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$ // Inference BN network with frozen // parameters
for $k = 1 \ldots K$ **do**
    // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_\mathcal{B} \equiv \mu_\mathcal{B}^{(k)}$, etc.
    Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_\mathcal{B}[\mu_\mathcal{B}]$$
$$\mathrm{Var}[x] \leftarrow \tfrac{m}{m-1}\mathrm{E}_\mathcal{B}[\sigma_\mathcal{B}^2]$$

    In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$
end for

Algorithm 2: Training a Batch-Normalized Network

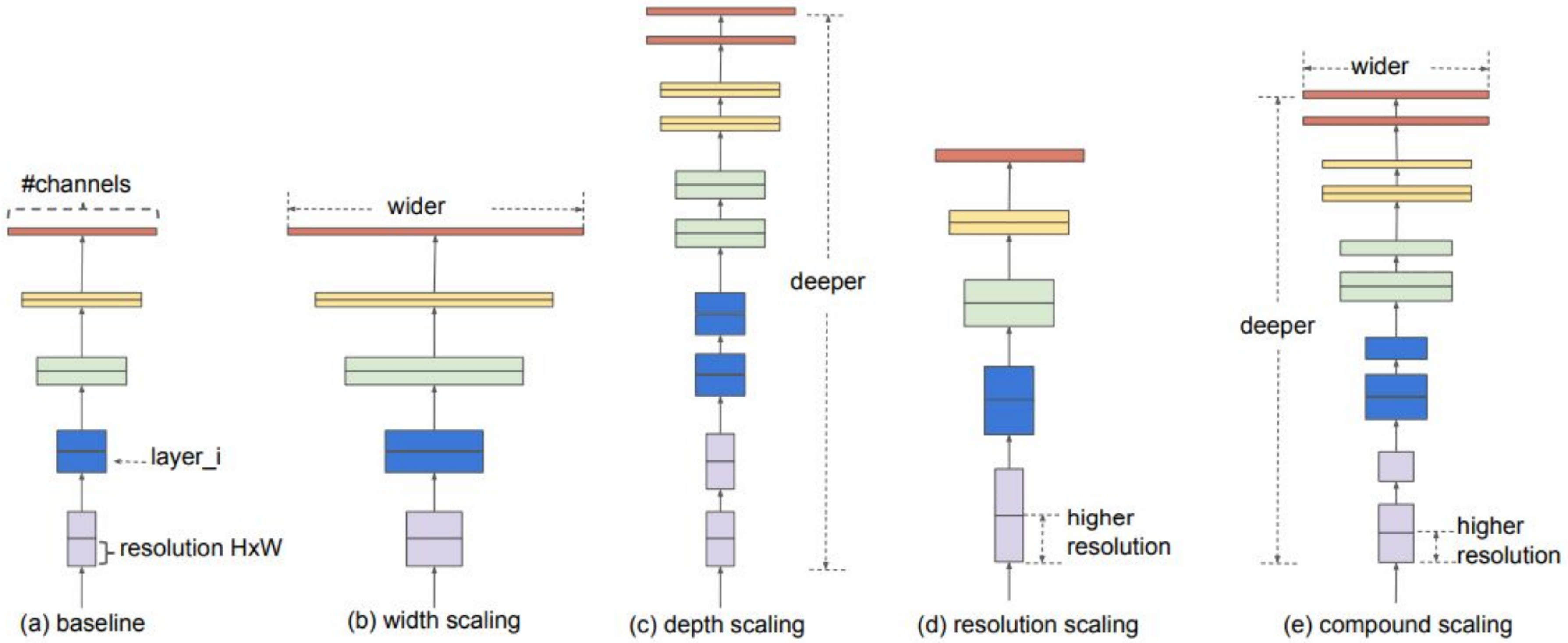
EfficientNet B0



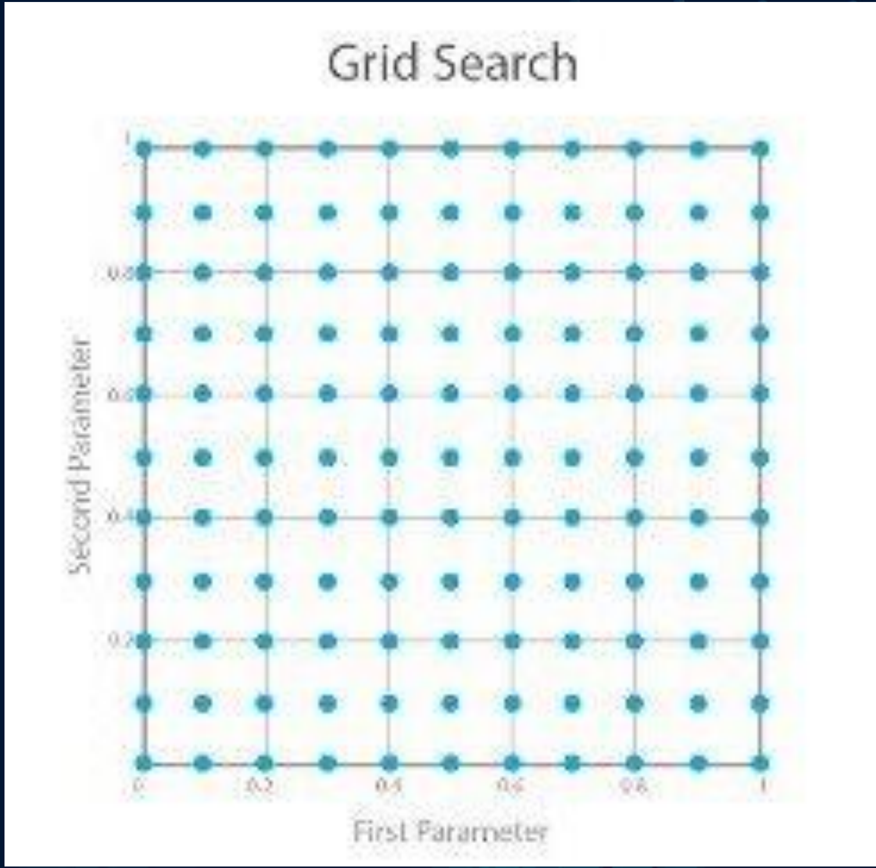
	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

† Not plotted

EfficientNet B0



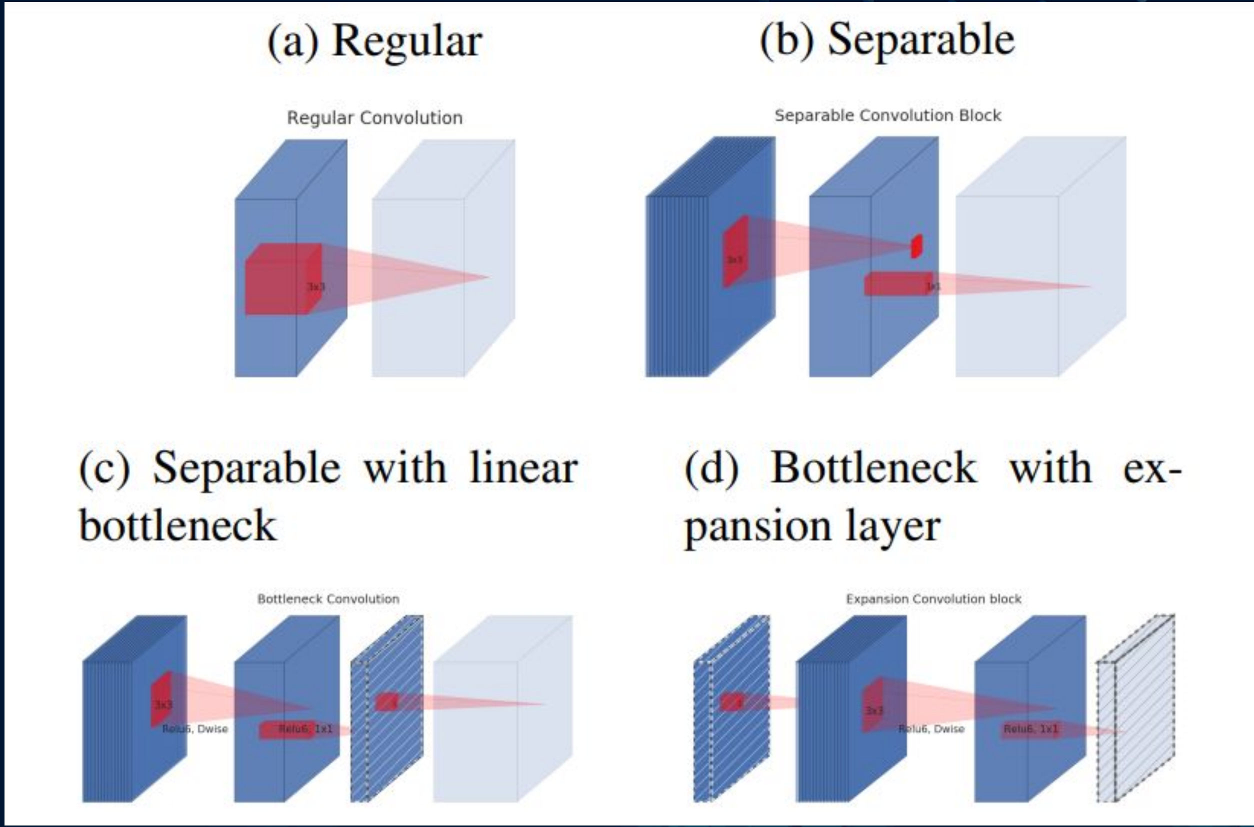
Grid Search



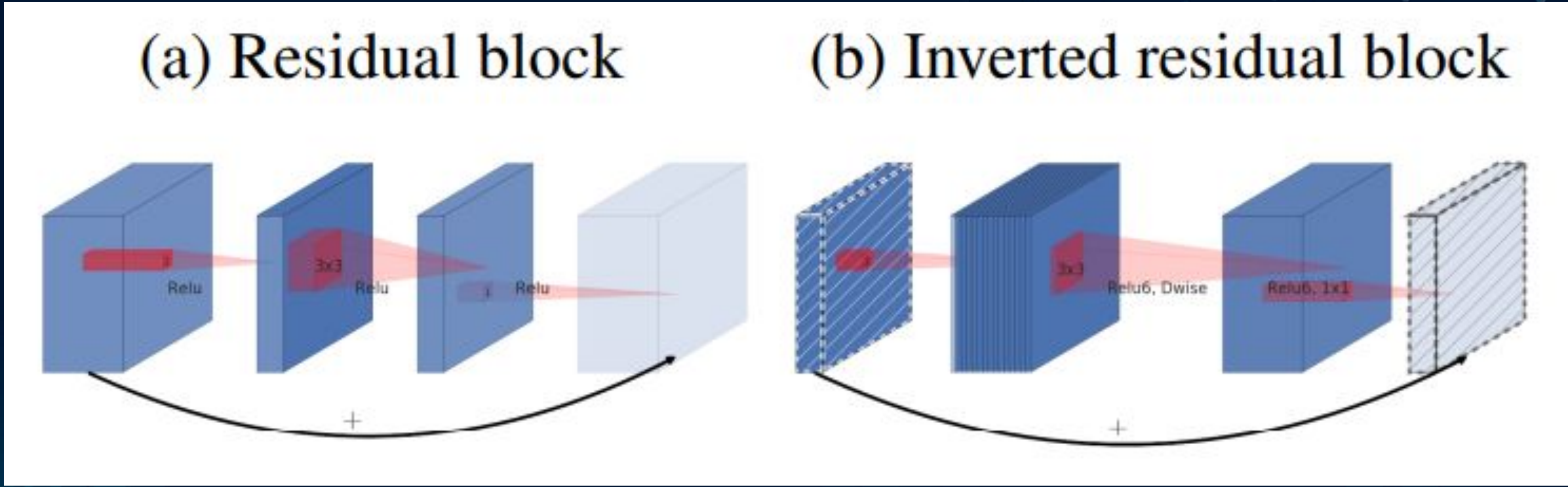
EfficientNet B0 Architecture

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

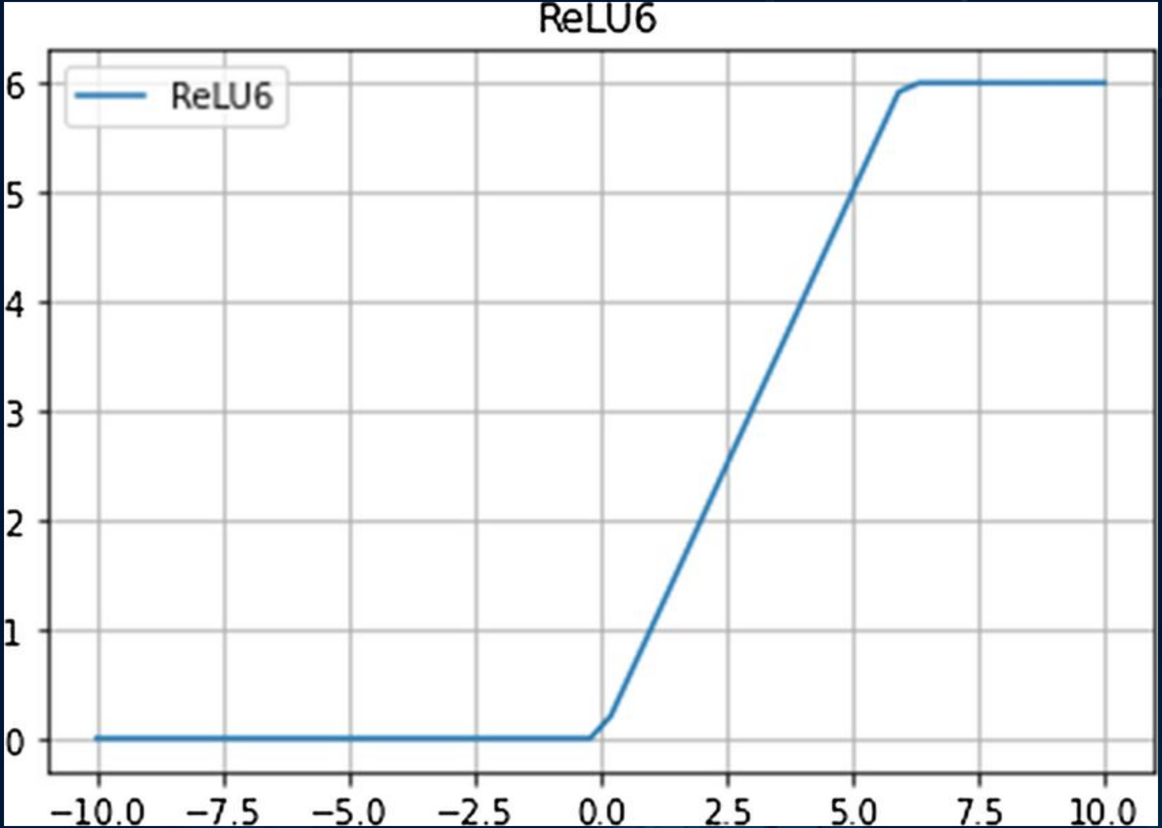
Inverted Residual Block



Inverted Residual Block



ReLU6



Compound Scaling

Baseline ResNet-50 (He et al., 2016)	4.1B	76.0%
Scale ResNet-50 by depth (d=4)	16.2B	78.1%
Scale ResNet-50 by width (w=2)	14.7B	77.7%
Scale ResNet-50 by resolution (r=2)	16.4B	77.5%
ResNet-50 compound scale	**16.7B**	**78.8%**

Inception (GoogleNet)

Inception v3 Architecture

type	patch size/stride or remarks	input size
conv	$3\times3/2$	$299\times299\times3$
conv	$3\times3/1$	$149\times149\times32$
conv padded	$3\times3/1$	$147\times147\times32$
pool	$3\times3/2$	$147\times147\times64$
conv	$3\times3/1$	$73\times73\times64$
conv	$3\times3/2$	$71\times71\times80$
conv	$3\times3/1$	$35\times35\times192$
$3\times$Inception	As in figure 5	$35\times35\times288$
$5\times$Inception	As in figure 6	$17\times17\times768$
$2\times$Inception	As in figure 7	$8\times8\times1280$
pool	8×8	$8\times8\times2048$
linear	logits	$1\times1\times2048$
softmax	classifier	$1\times1\times1000$

Figure 5

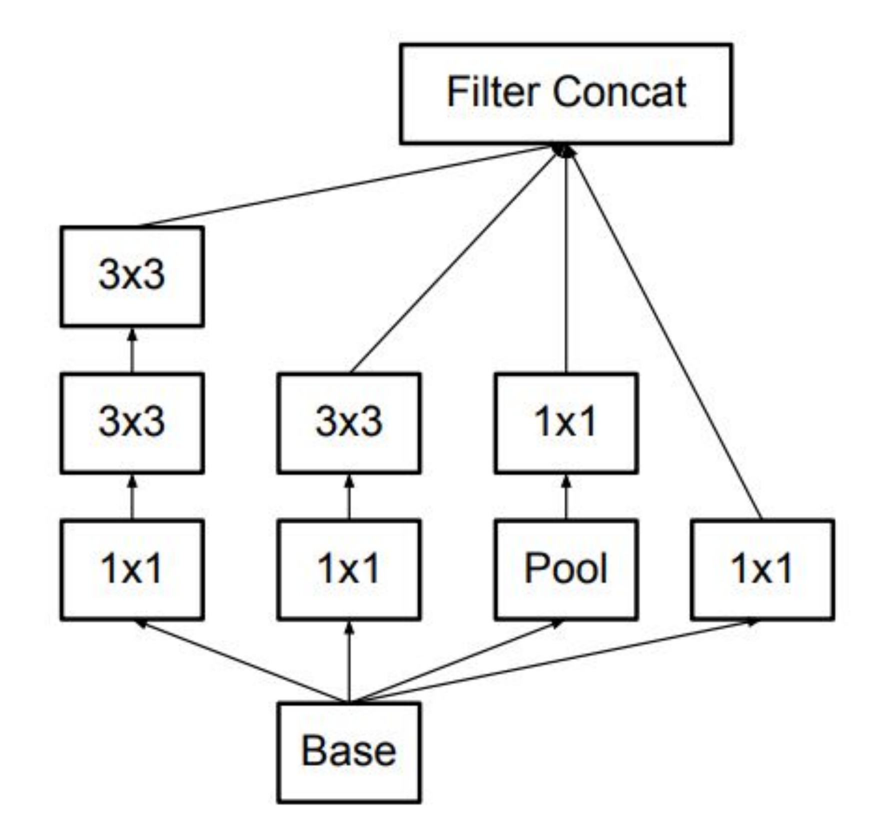
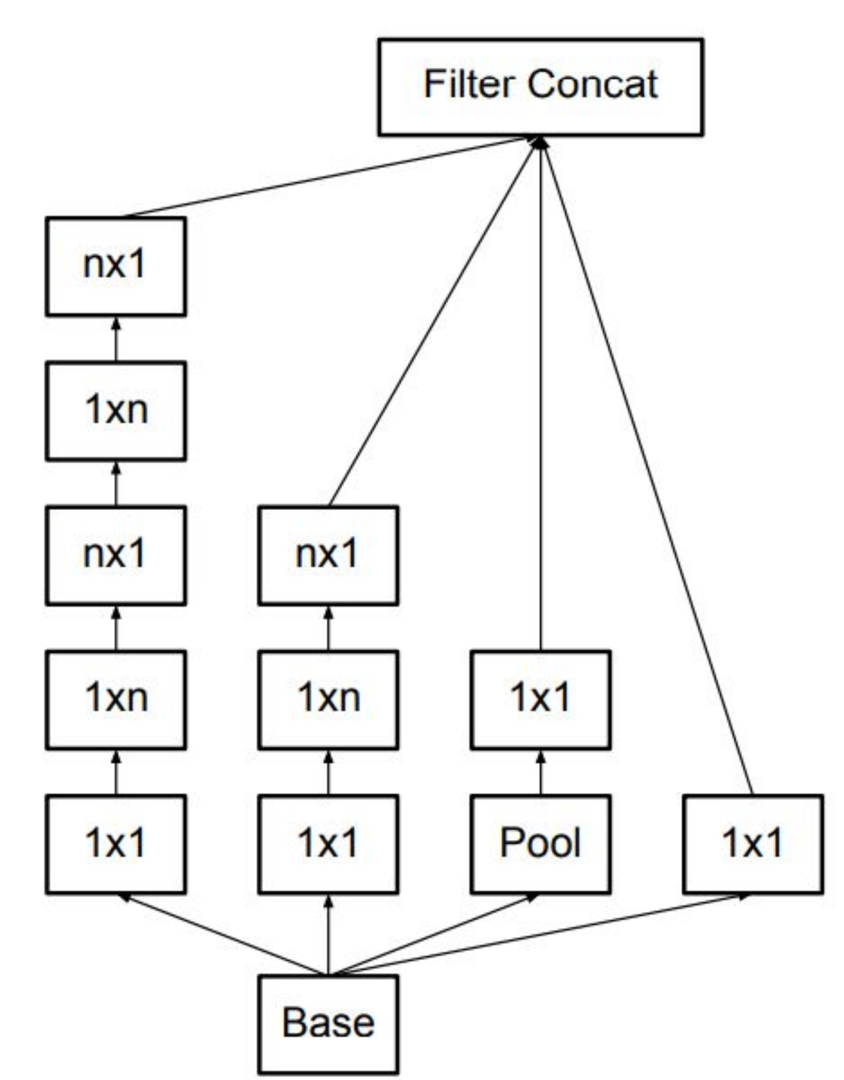
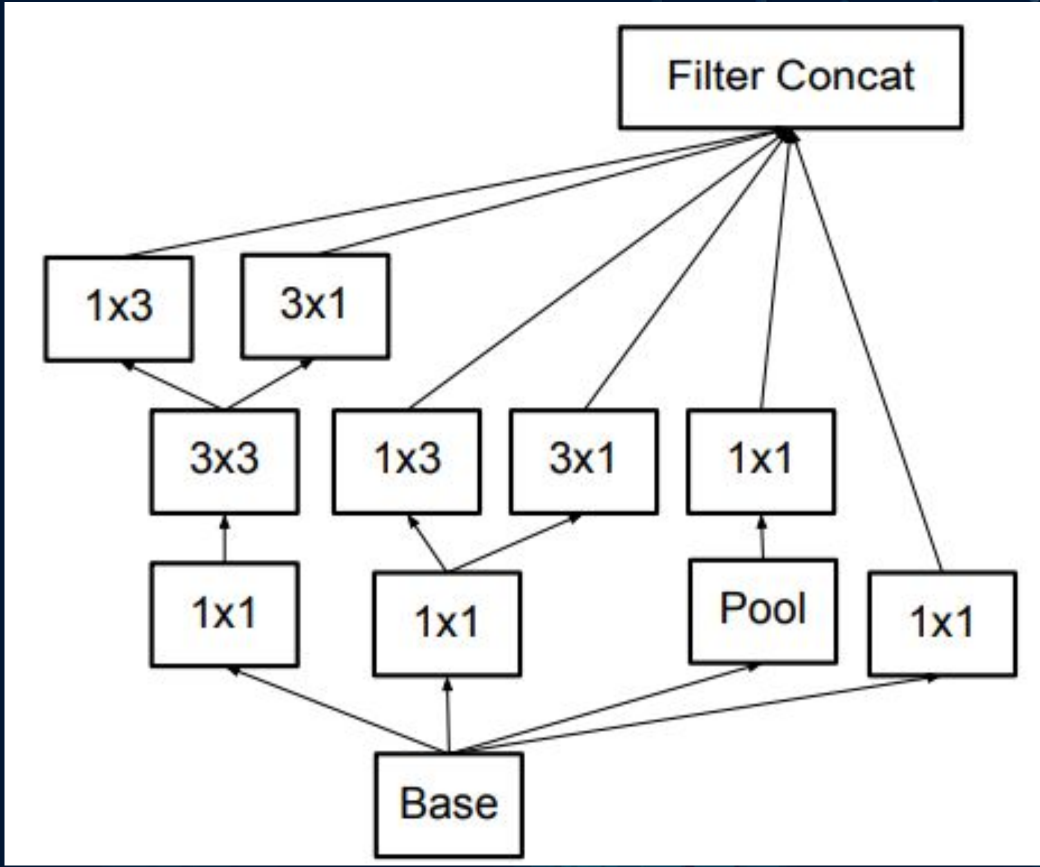


Figure 6

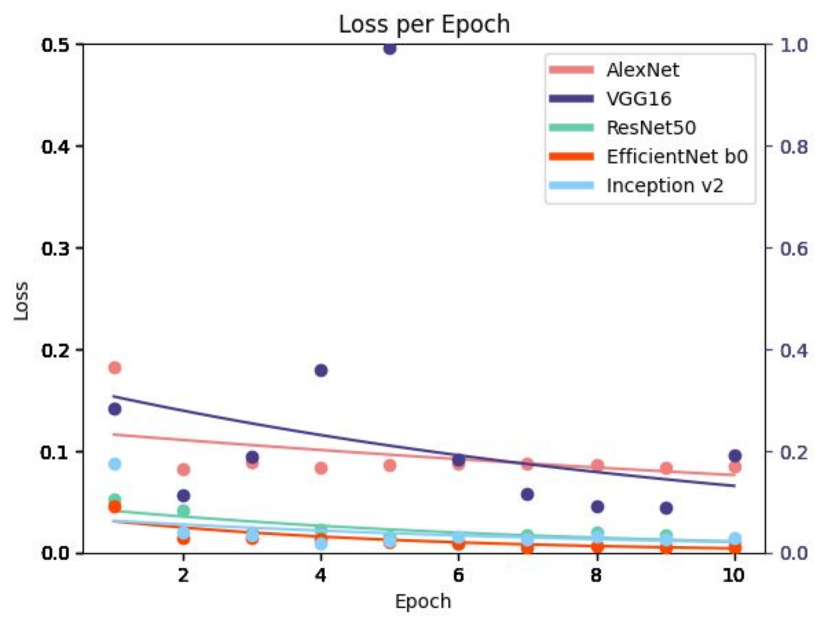


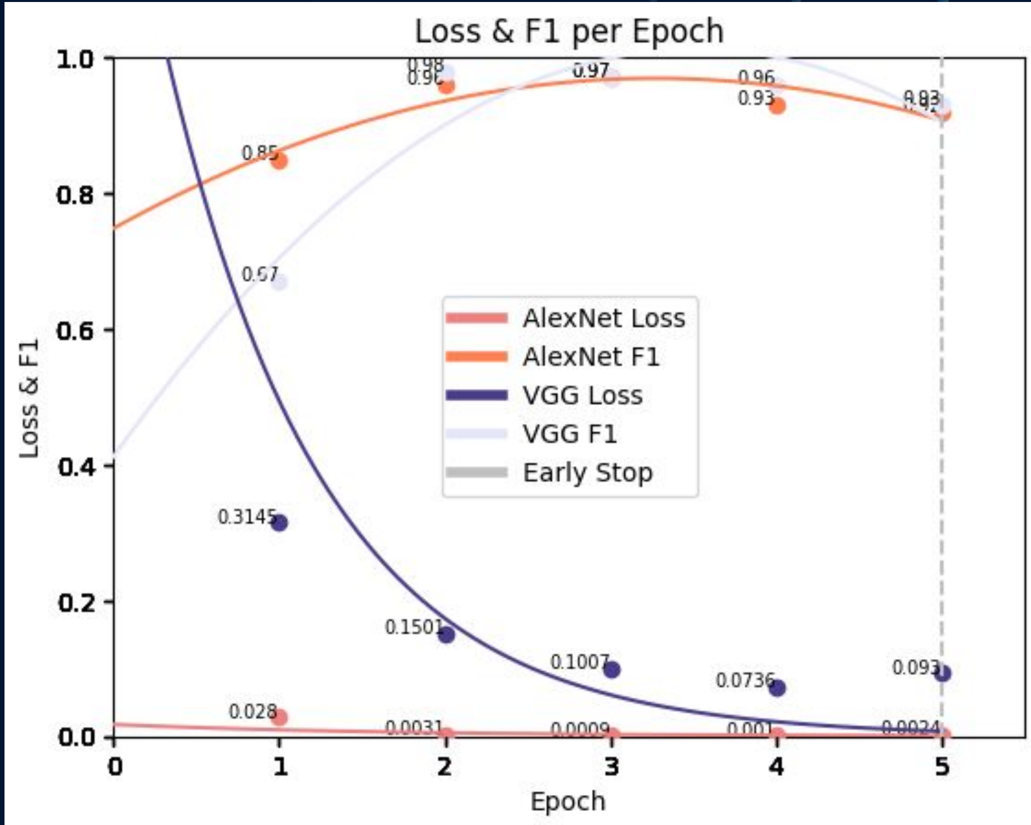
Filter Concat
nx1
1xn
nx1
nx1
1xn
1xn
1x1
1x1
1x1
Pool
1x1
Base

Figure 7



Loss Per Epoch w/ Curve Fit for All Models



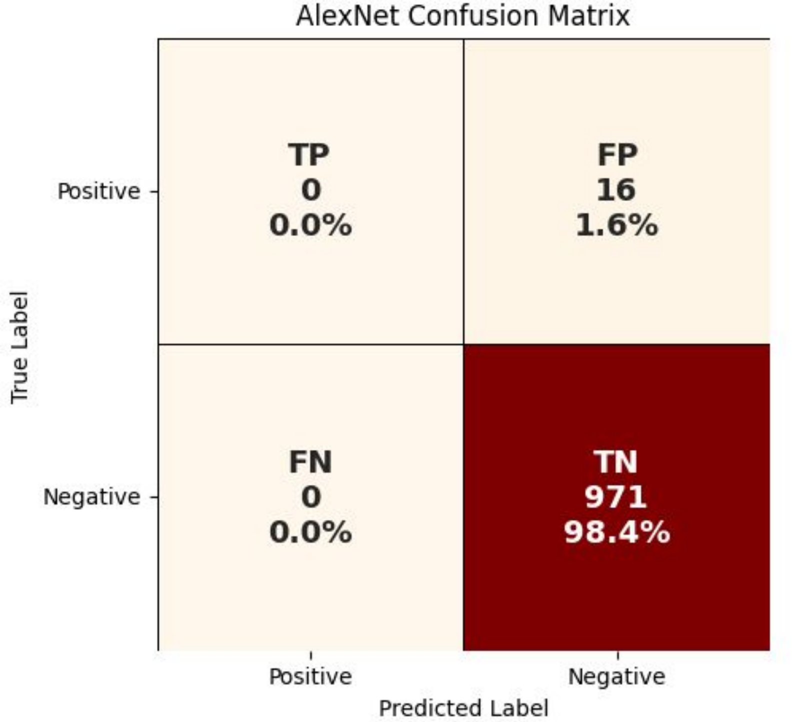


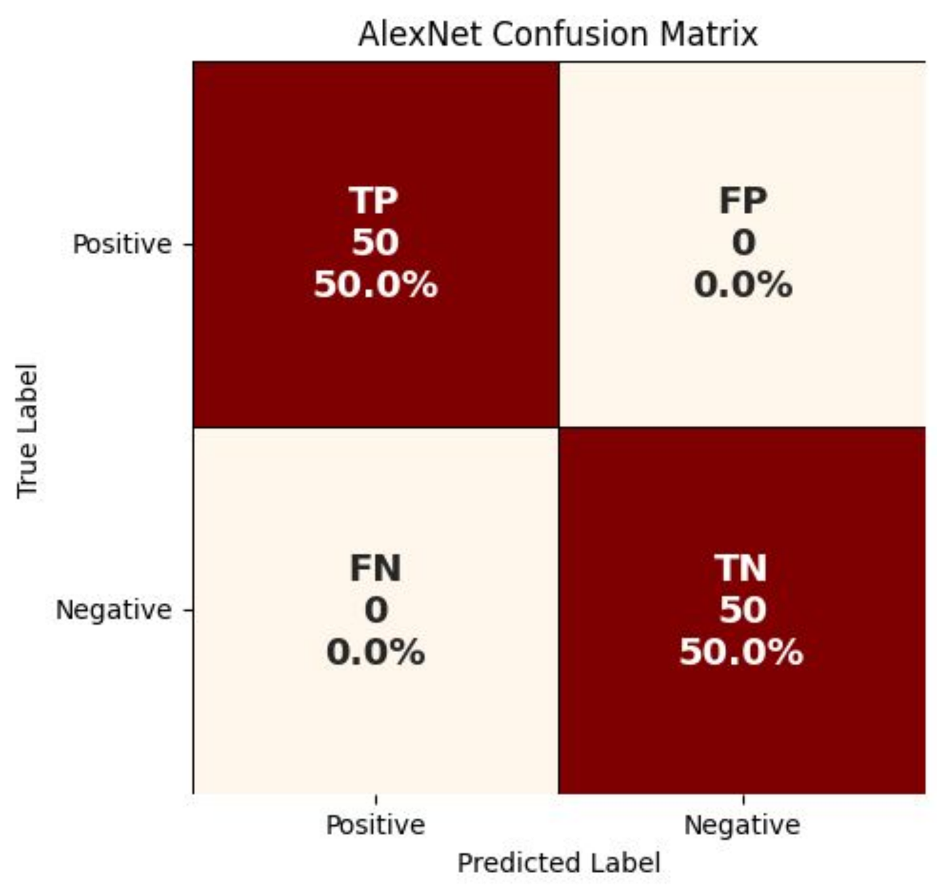
Pre-Trained Models Used & Metrics

	Metric	AlexNet	VGG16	ResNet50	EfficientNet b0	Inception v3
0	Accuracy	0.98	0.98	0.98	0.99	0.99
1	Precision	0.98	0.98	1.00	0.99	0.99
2	F1	0.99	0.99	0.99	1.00	1.00
3	ROC AUC	0.50	0.40	0.96	1.00	0.97
4	Runtime (sec)	173.46	676.63	484.59	301.84	662.06

Metric	AlexNet	VGG16	ResNet50	EfficeintNet b0	Inception v3
Accuracy	1	0.93	0.89	0.83	0.93
Precision	1	0.89	0.83	0.75	0.88
F1	1	0.93	0.9	0.85	0.93
AUC of ROC	1	0.94	0.94	1	1
Runtime(sec)	118.23	244.61	596.17	358.95	724.81
Parameter #	61,100,840	138,357,544	25,557,032	5.3M	27,161,264

AlexNet Confusion Matrix

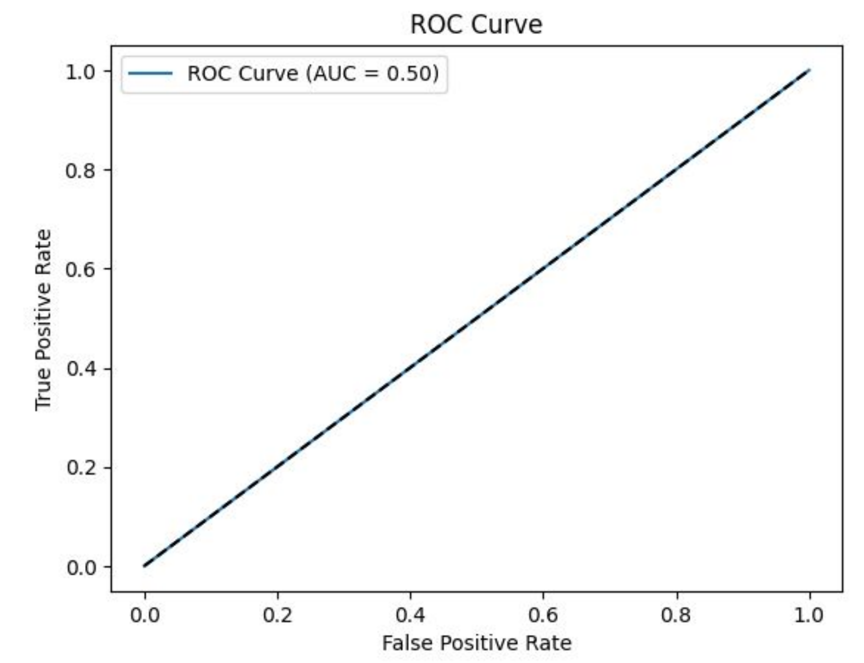


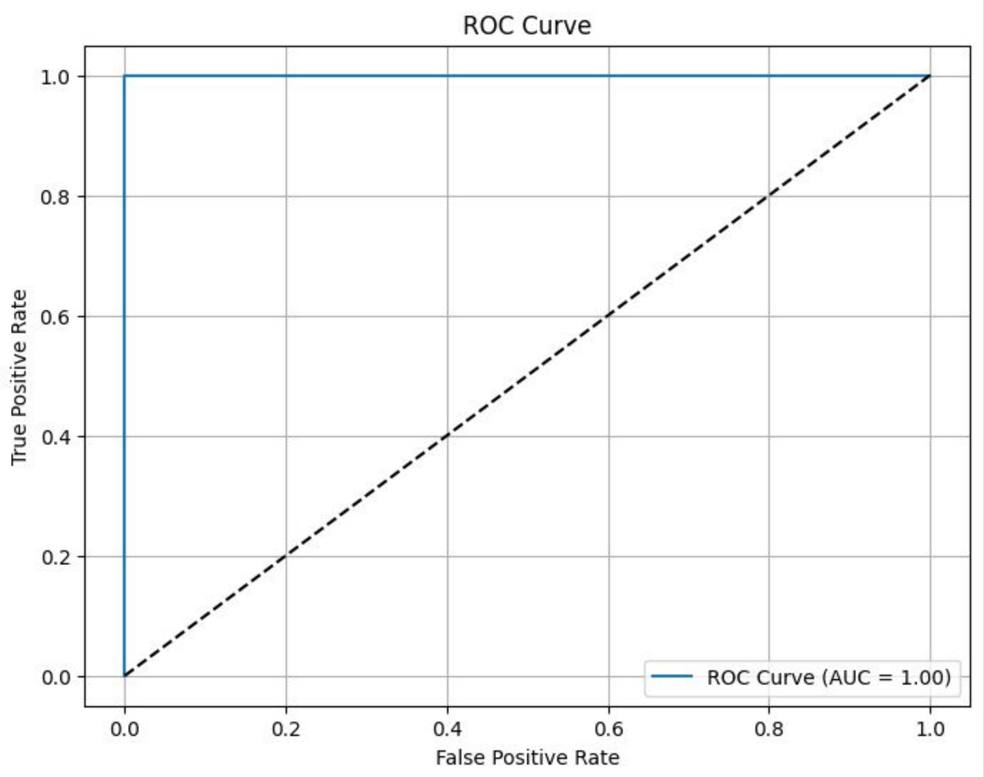


AlexNet ROC/ AUC Plot

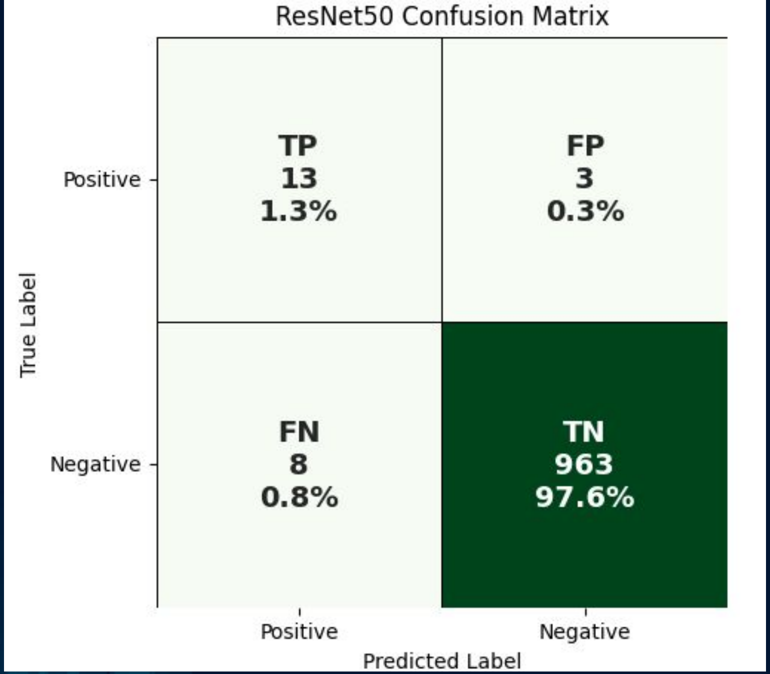
$$TPR = \frac{TP}{TP + FN}$$

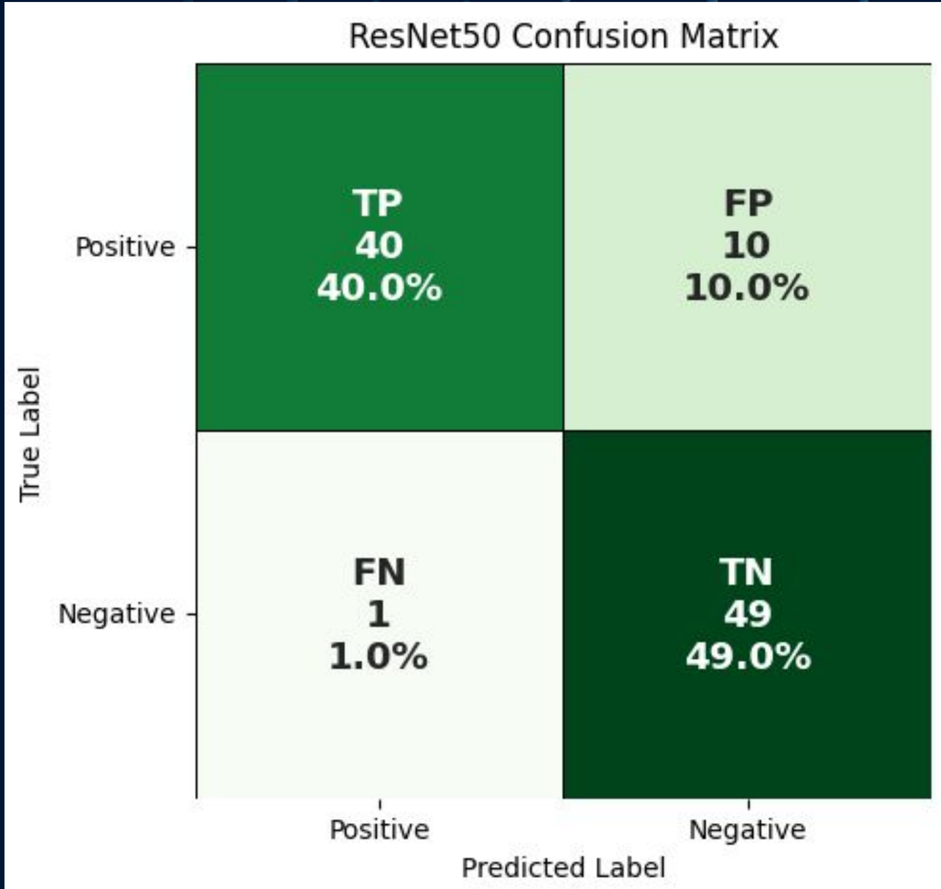
$$FPR = \frac{FP}{FP + TN}$$



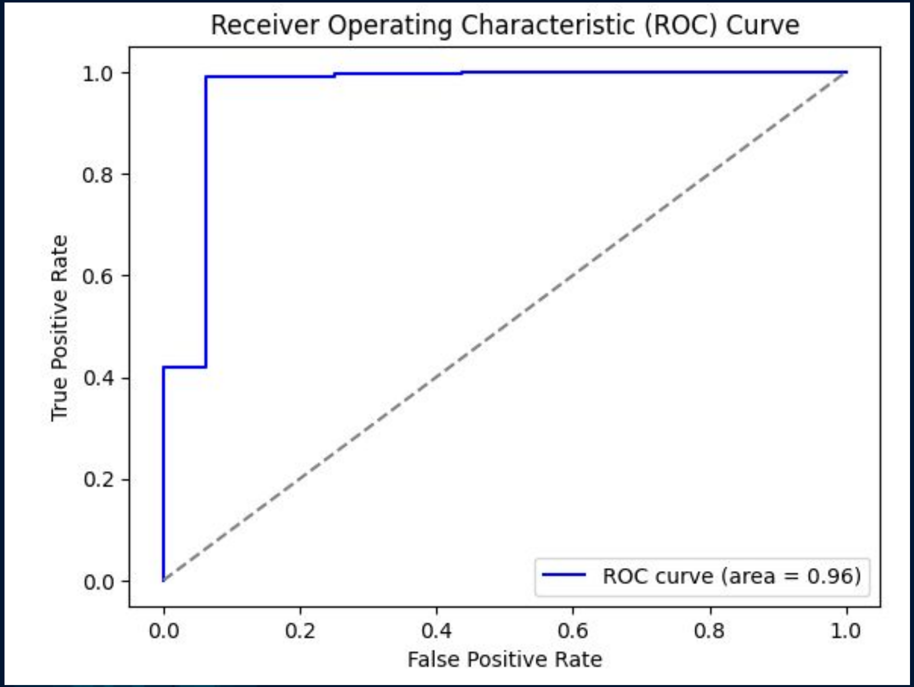


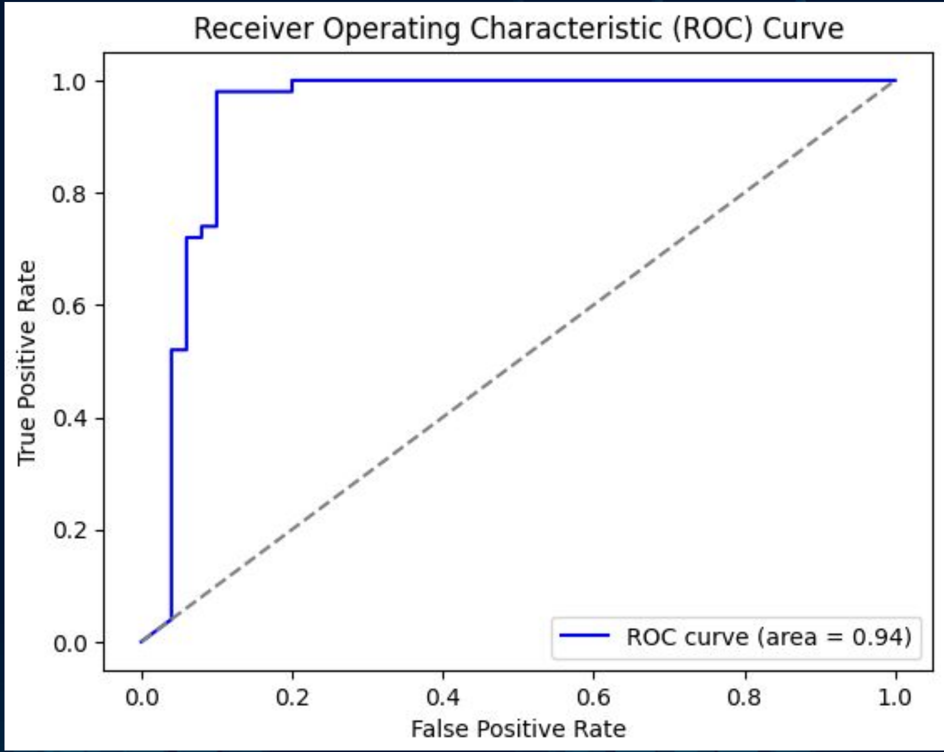
ResNet50 Confusion Matrix



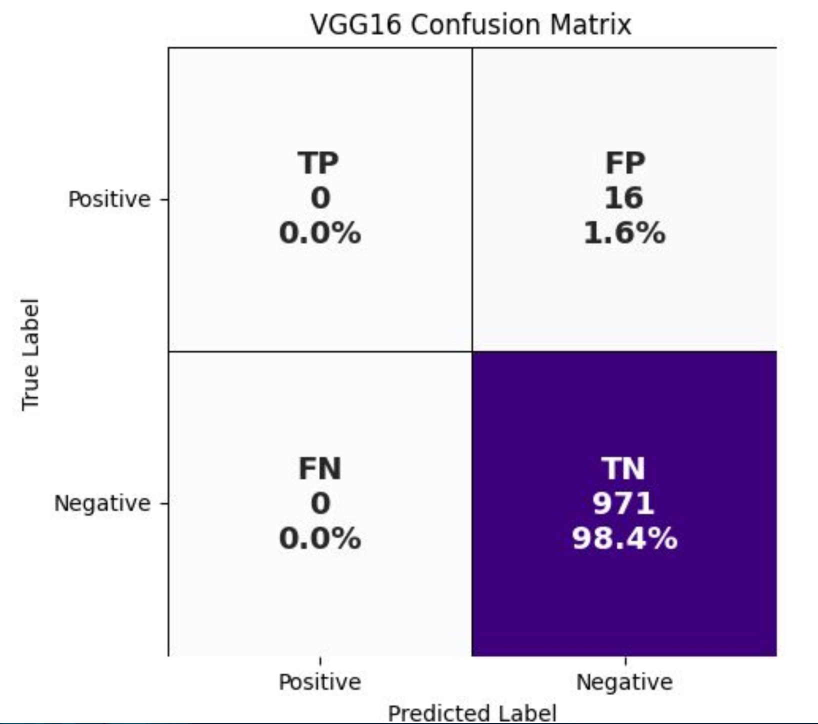


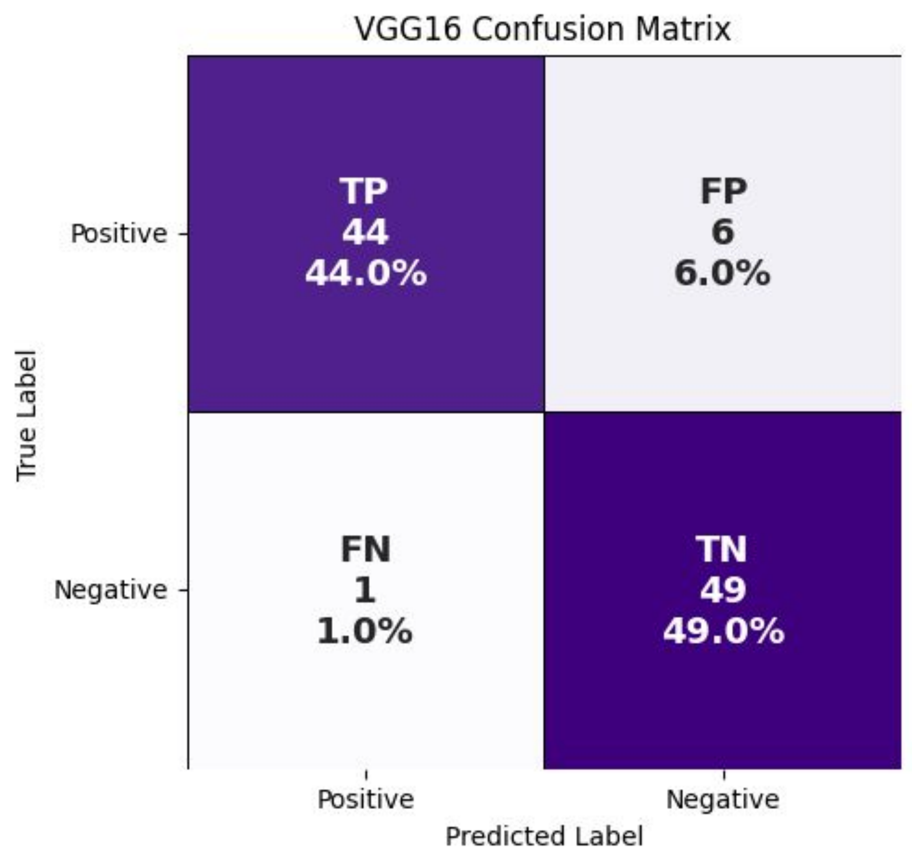
ResNet50 ROC/ AUC Plot



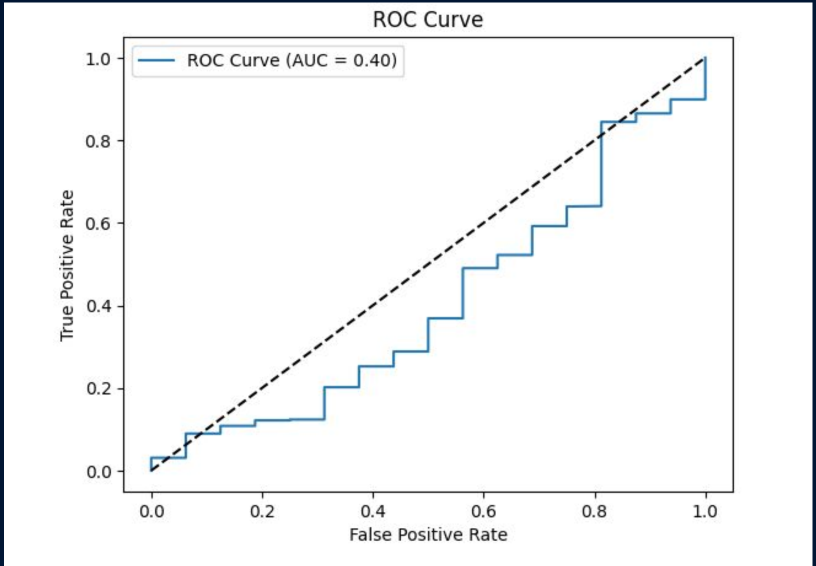


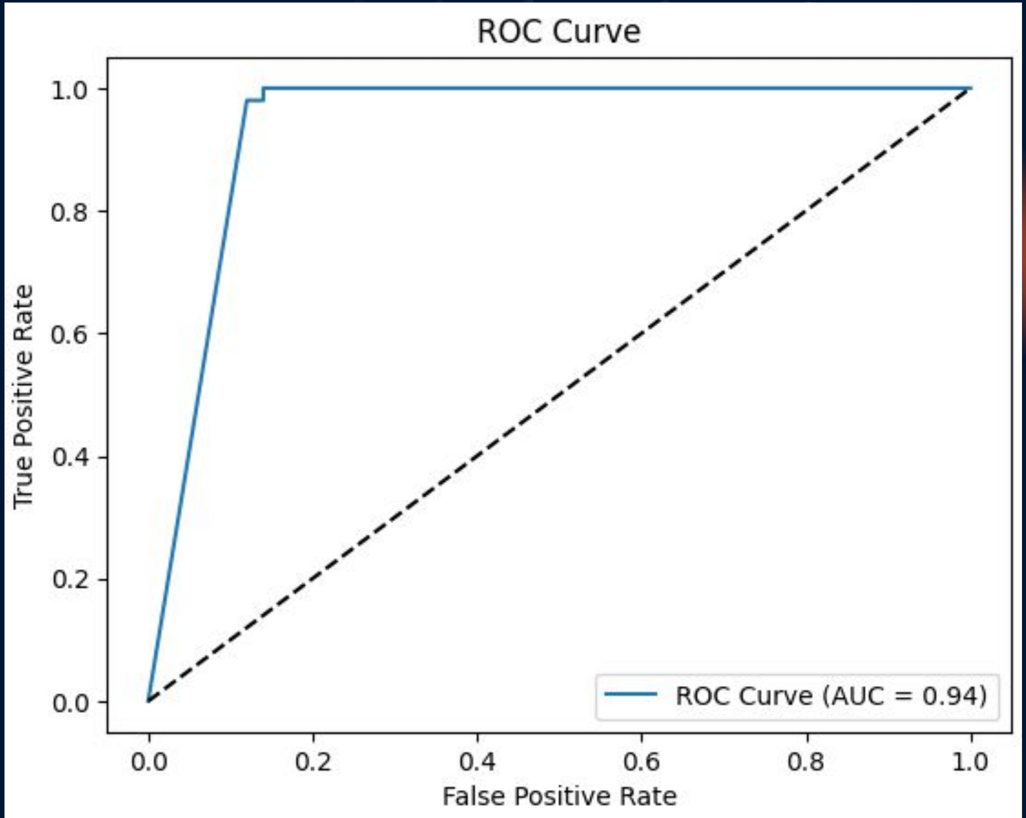
VGG16 Confusion Matrix



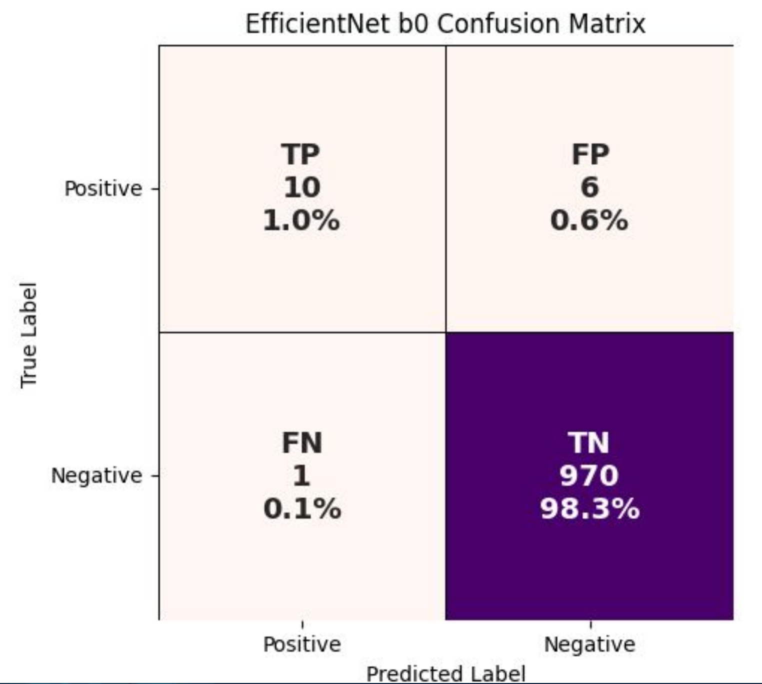


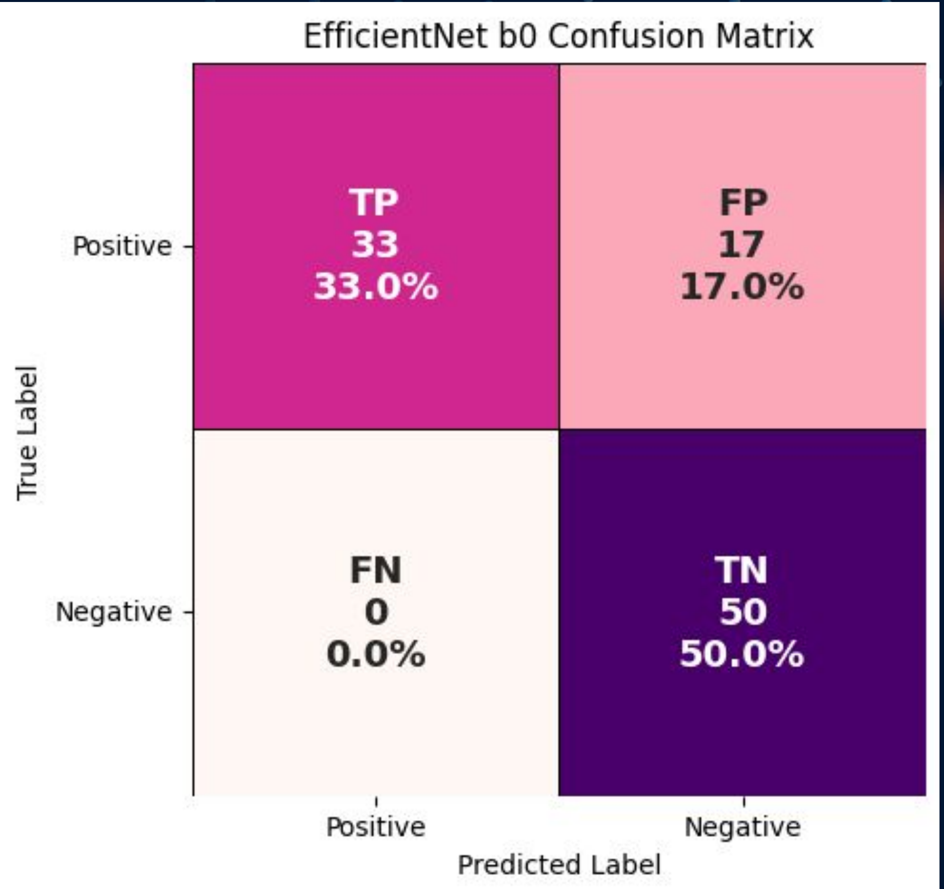
VGG16 ROC/ AUC Plot



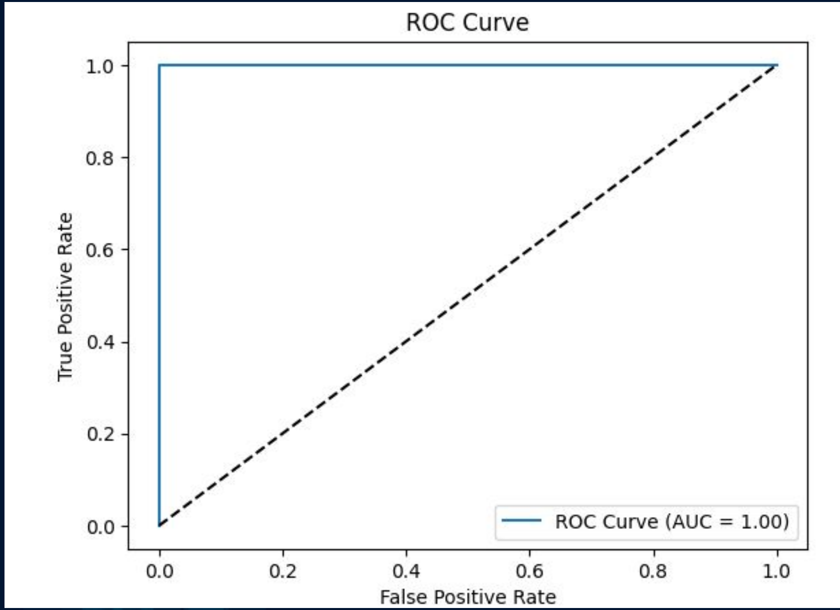


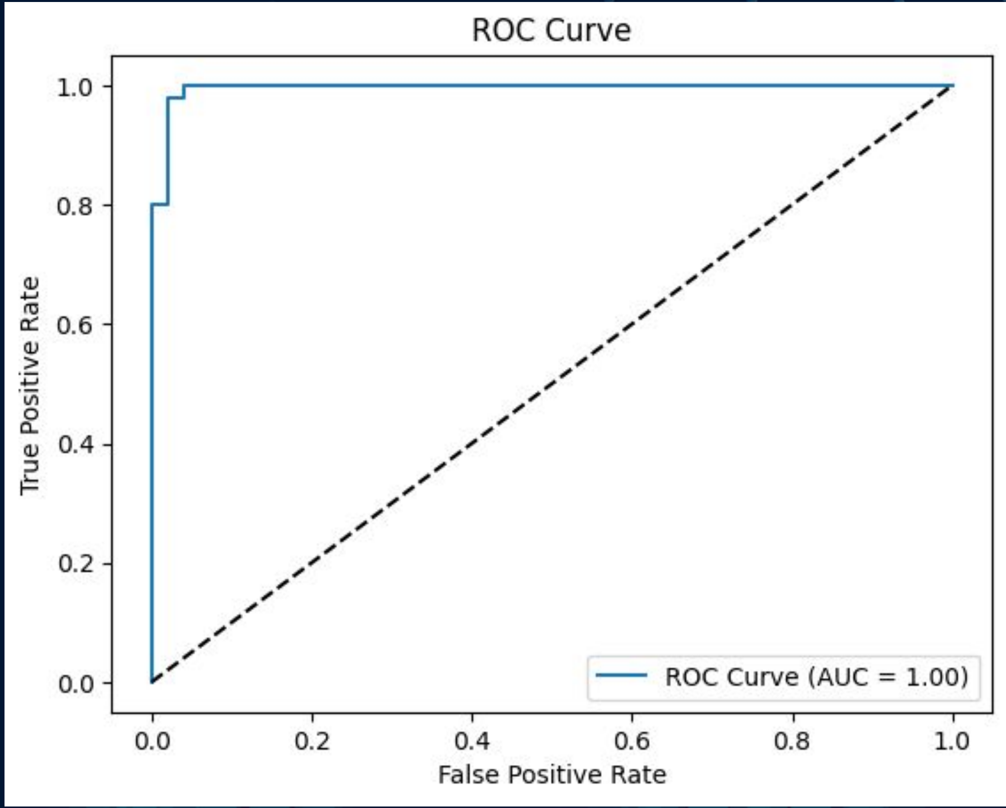
EfficientNet b0 Confusion Matrix



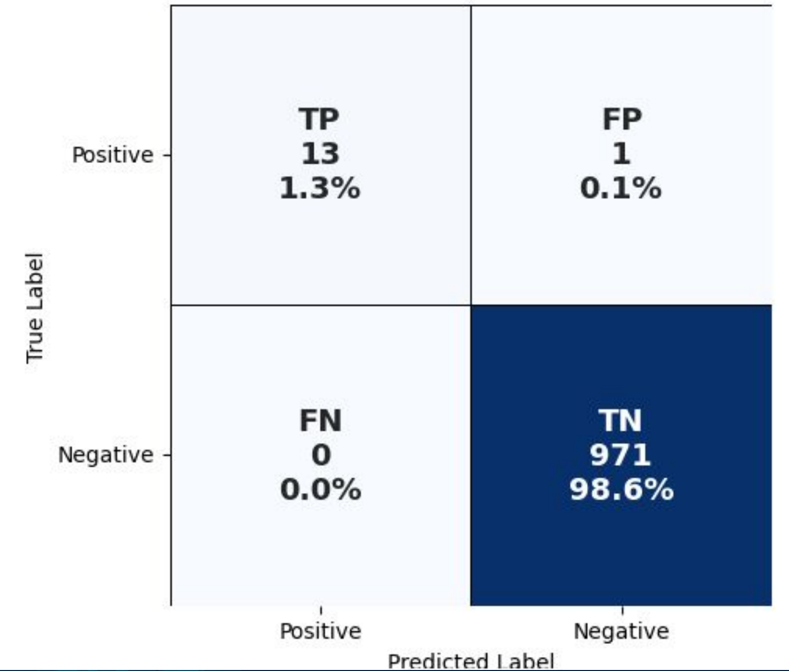


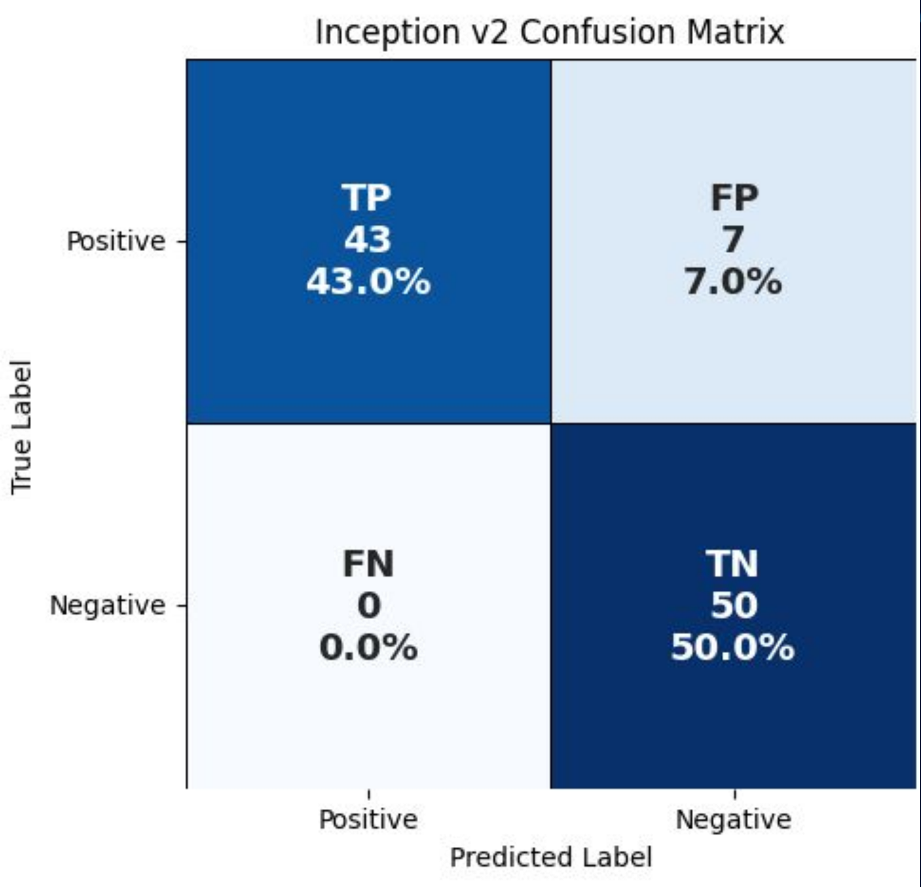
EfficientNet b0 ROC/ AUC Plot



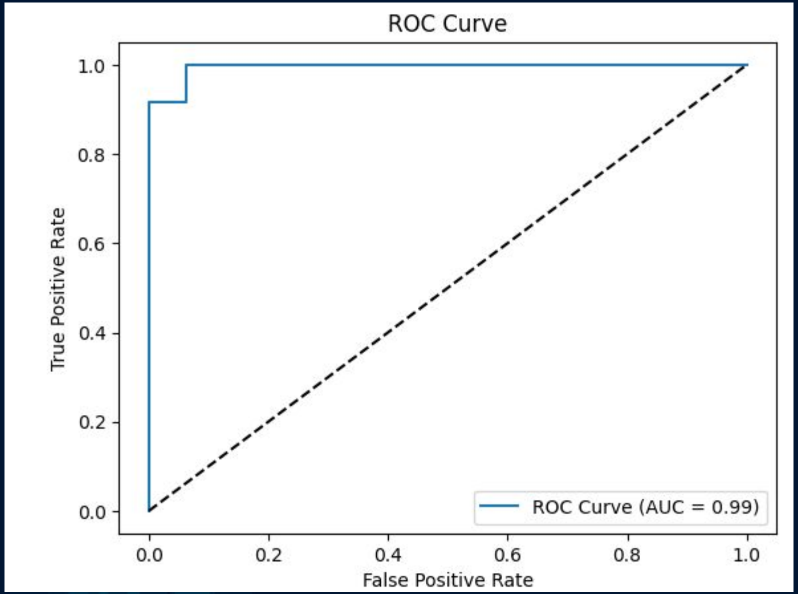


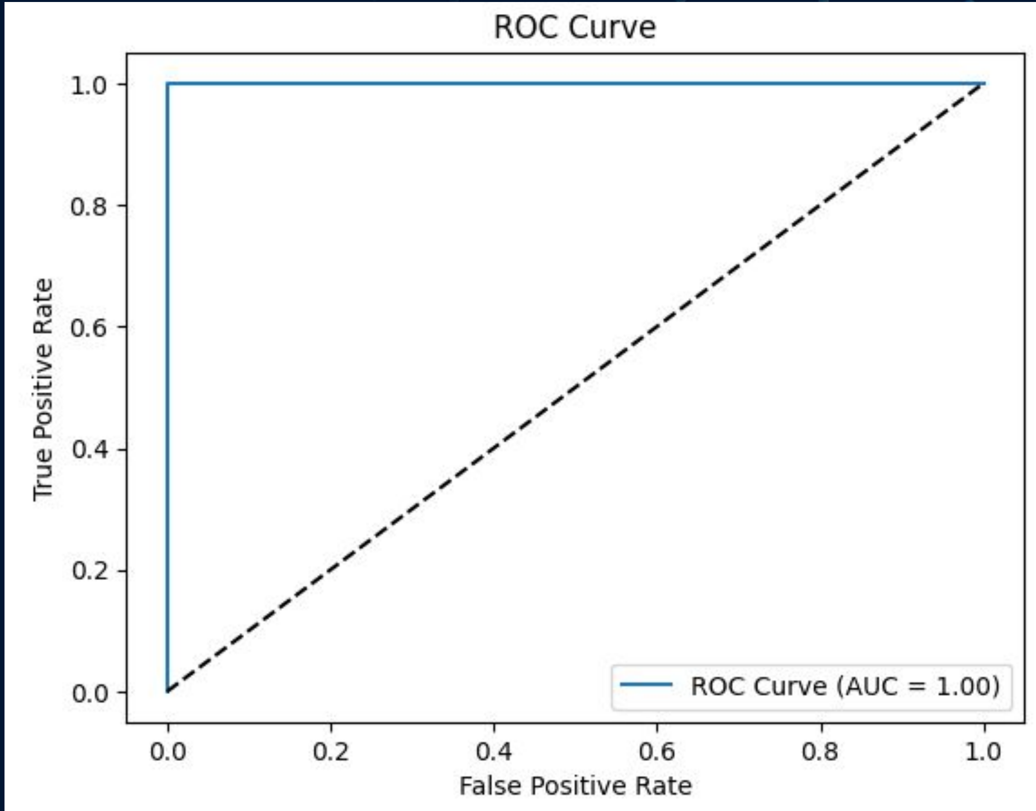
Inception v2 Confusion Matrix





Inception v2 ROC/ AUC Plot





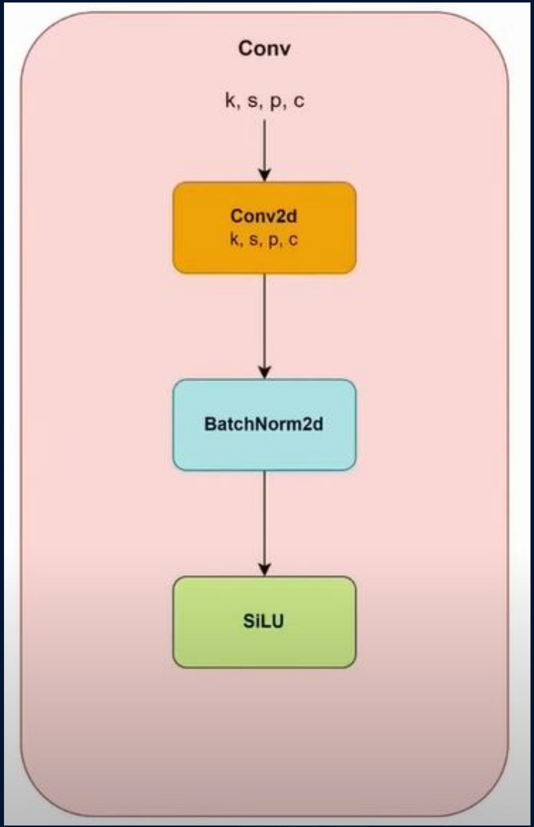
YOLO Colab Implementation

```
!pip install ultralytics
from ultralytics import YOLO
model = YOLO("yolov8n.pt")
# Train the model
model.train(data='graphene.yaml', epochs=10, imgsz=640)
```

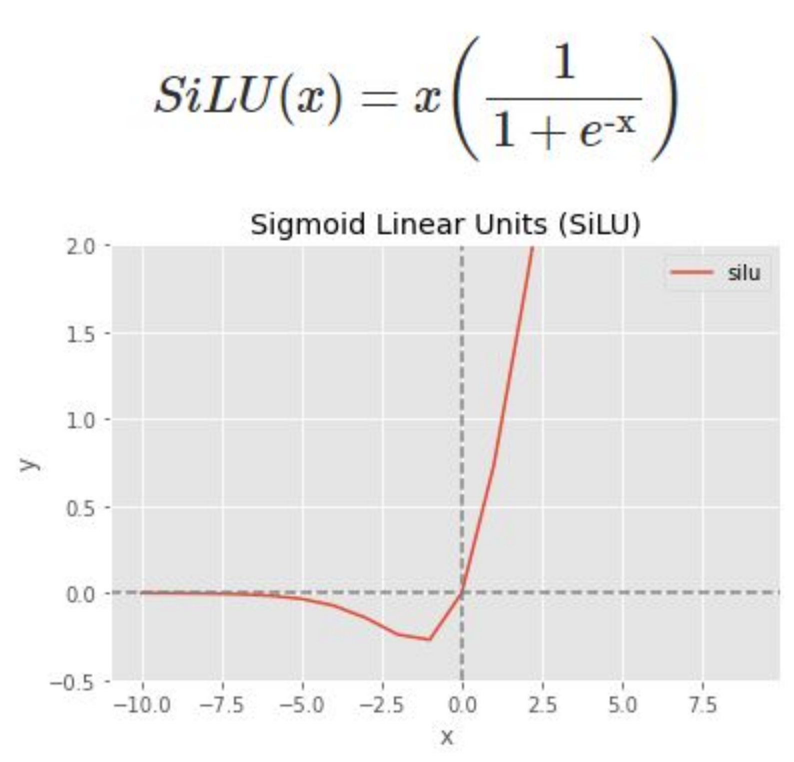
```
Data Structure
|__>train
        |__>images
        |__>labels (.txt)
|__>val
        |__>images
        |__>labels (.txt)
```

XX.

YOLOv8 Architecture

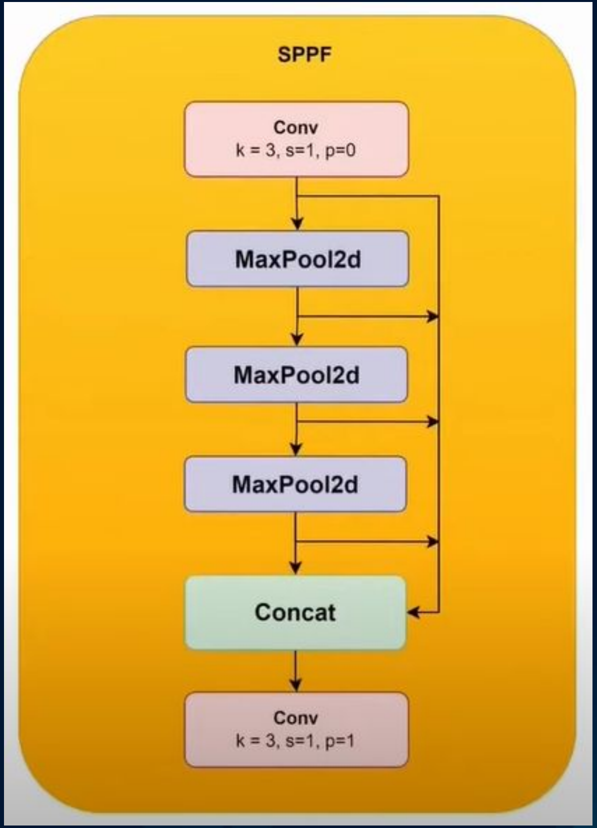


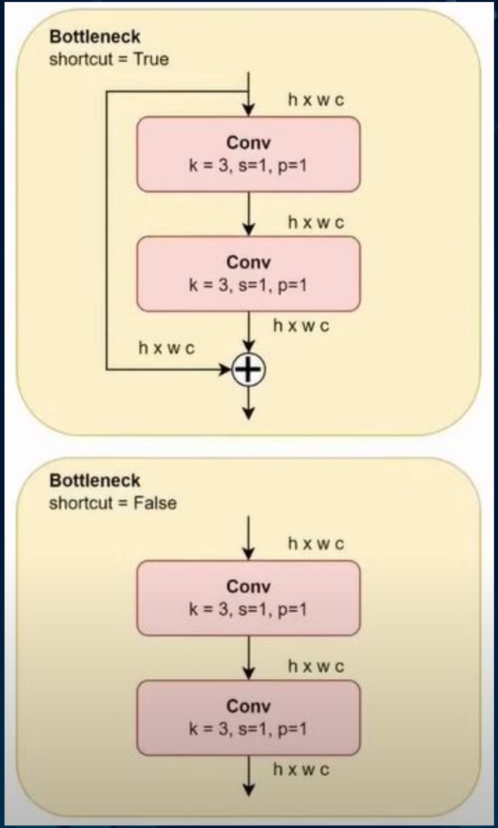
XXII.



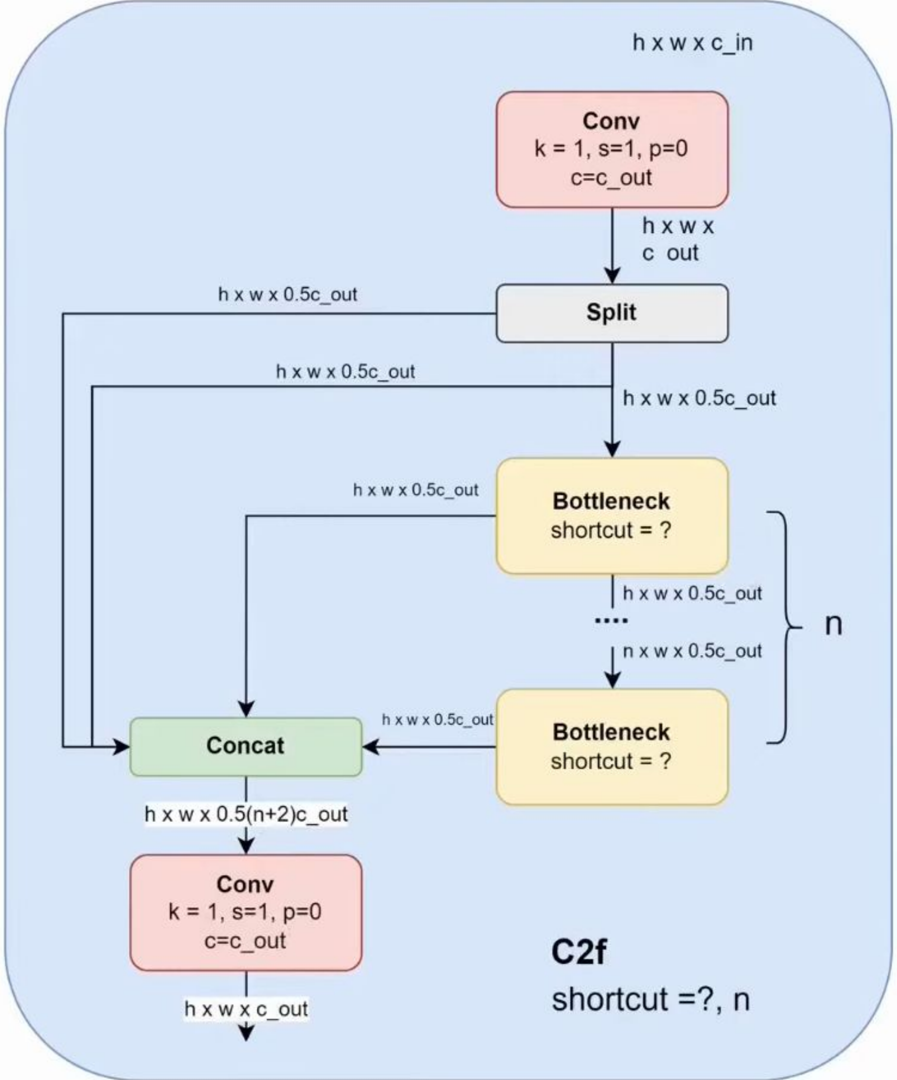
XXI.

YOLOv8 Architecture

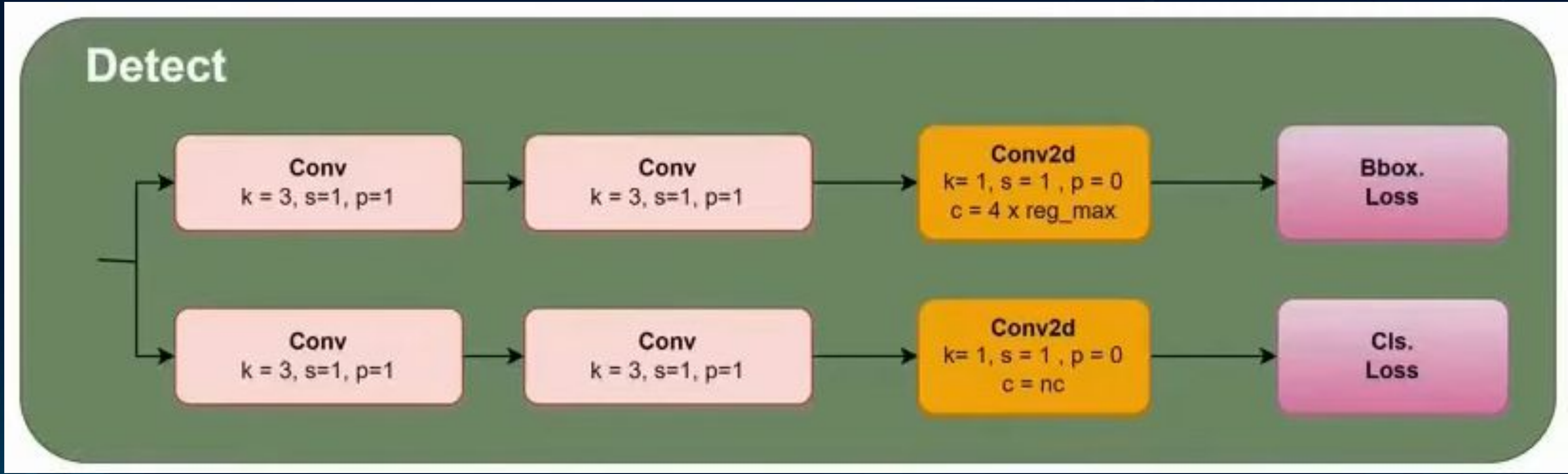




YOLOv8 Architecture

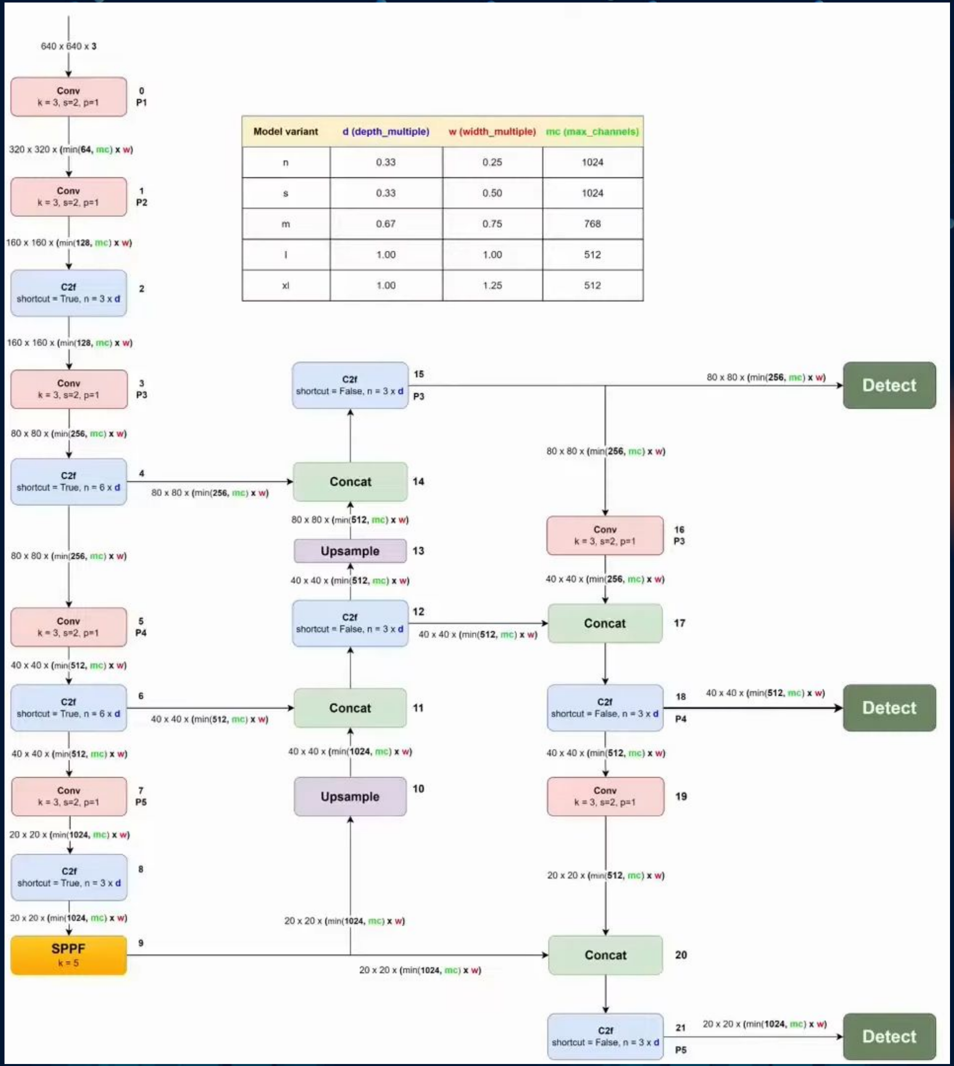


YOLOv8 Architecture

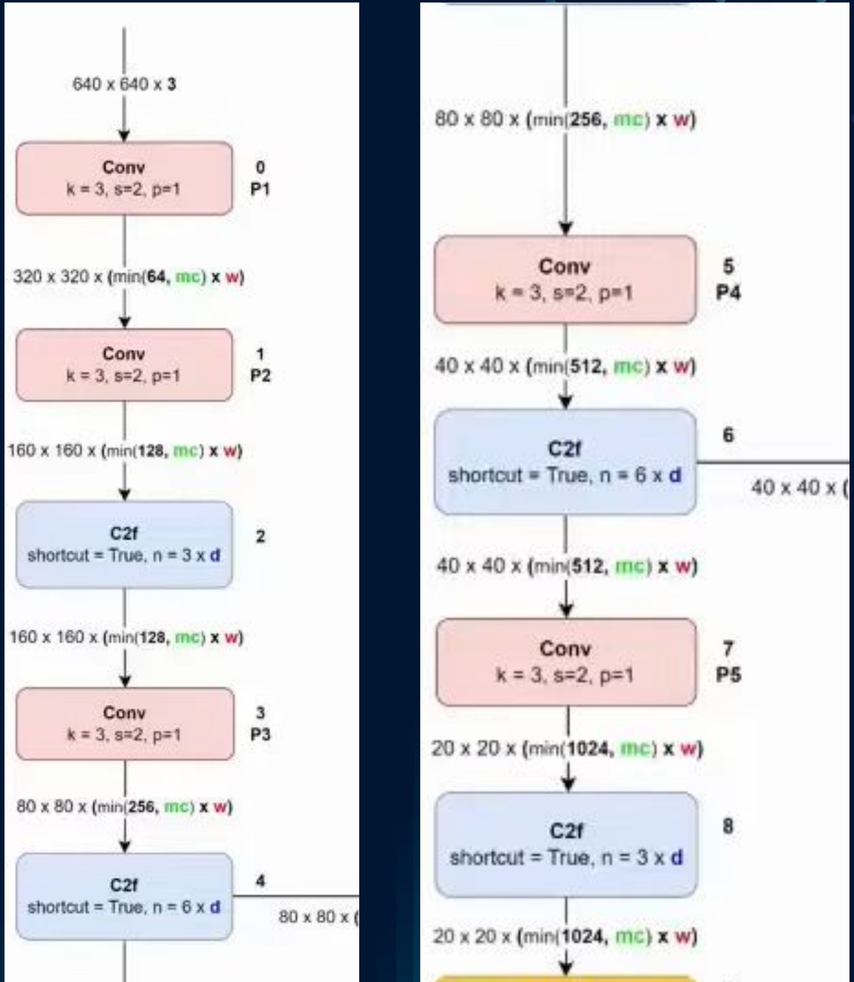


YOLOv8 Architecture

Model variant	d (depth_multiple)	w (width_multiple)	mc (max_channels)
n	0.33	0.25	1024
s	0.33	0.50	1024
m	0.67	0.75	768
l	1.00	1.00	512
xl	1.00	1.25	512

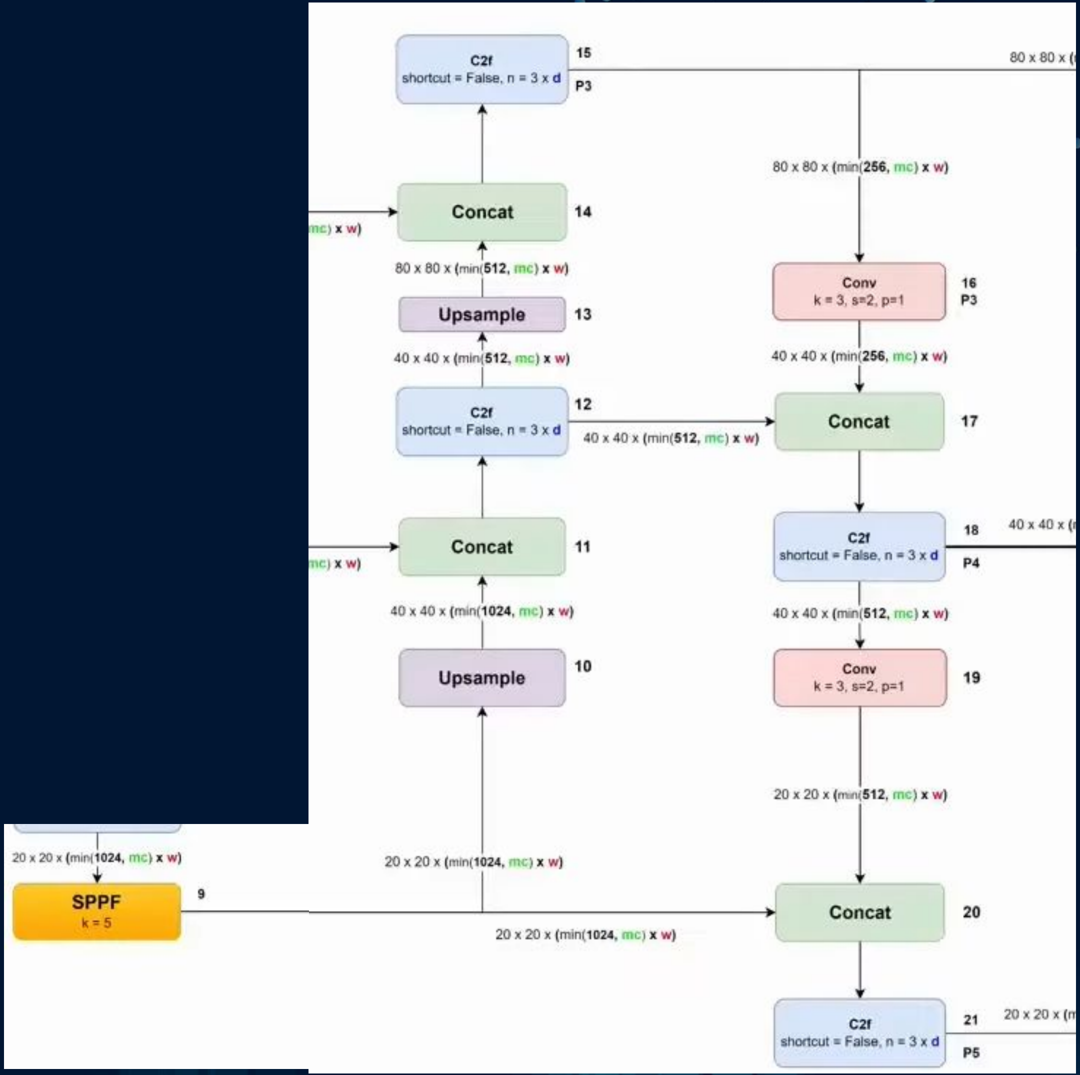


Backbone

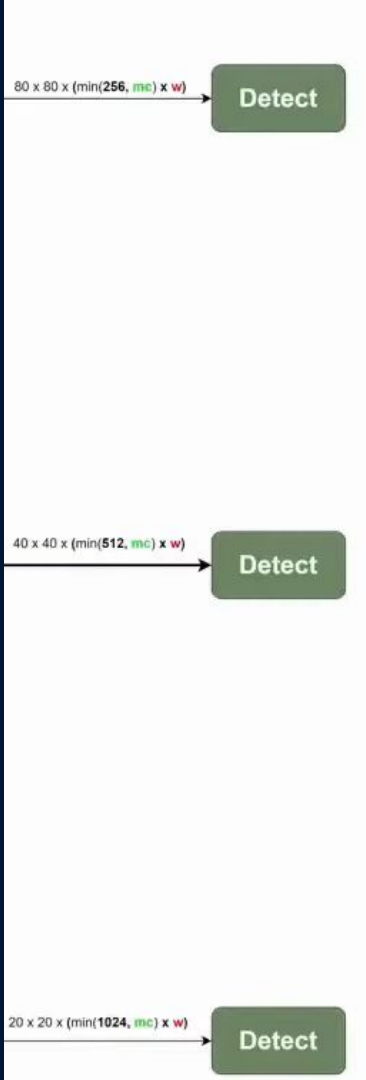


640 x 640 x 3
Conv
k = 3, s=2, p=1
0
P1
320 x 320 x (min(64, mc) x w)
Conv
k = 3, s=2, p=1
1
P2
160 x 160 x (min(128, mc) x w)
C2f
shortcut = True, n = 3 x d
2
160 x 160 x (min(128, mc) x w)
Conv
k = 3, s=2, p=1
3
P3
80 x 80 x (min(256, mc) x w)
C2f
shortcut = True, n = 6 x d
4
80 x 80 x (
80 x 80 x (min(256, mc) x w)
Conv
k = 3, s=2, p=1
5
P4
40 x 40 x (min(512, mc) x w)
C2f
shortcut = True, n = 6 x d
6
40 x 40 x (
40 x 40 x (min(512, mc) x w)
Conv
k = 3, s=2, p=1
7
P5
20 x 20 x (min(1024, mc) x w)
C2f
shortcut = True, n = 3 x d
8
20 x 20 x (min(1024, mc) x w)

Neck


C2f
shortcut = False, n = 3 x d
15
P3
80 x 80 x (min(256, mc) x w)
Concat
14
mc) x w)
80 x 80 x (min(512, mc) x w)
Upsample
13
40 x 40 x (min(512, mc) x w)
C2f
shortcut = False, n = 3 x d
12
40 x 40 x (min(512, mc) x w)
Conv
k = 3, s=2, p=1
16
P3
40 x 40 x (min(256, mc) x w)
Concat
17
Concat
11
mc) x w)
40 x 40 x (min(1024, mc) x w)
C2f
shortcut = False, n = 3 x d
18
P4
40 x 40 x (min(512, mc) x w)
Upsample
10
Conv
k = 3, s=2, p=1
19
20 x 20 x (min(512, mc) x w)
20 x 20 x (min(1024, mc) x w)
20 x 20 x (min(1024, mc) x w)
SPPF
k = 5
9
Concat
20
20 x 20 x (min(1024, mc) x w)
C2f
shortcut = False, n = 3 x d
21
P5

Head



80 x 80 x (min(256, mc) x w)
Detect
40 x 40 x (min(512, mc) x w)
Detect
20 x 20 x (min(1024, mc) x w)
Detect

AdamW Optimizer Psuedocode (Algo. 1 is S

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$

initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$

repeat

  $t \leftarrow t + 1$

  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient

  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$

  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise

  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$

  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t

  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t

  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts

  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$

until *stopping criterion is met*

return optimized parameters $\boldsymbol{\theta}_t$

Acknowledgements

- Abhay Narayan Pasupathy Research Group https://anp-lab.physics.columbia.edu/
- Reproducibility: https://github.com/56sarager/Graphene-Classification , https://drive.google.com/drive/folders/1CJKs_0wGzySyGWV78xK-VB7FZPO_-qDO
- Johannes Geurs https://www.nature.com/articles/s41598-018-23922-1 (paper develops a program for identifying graphene flake from microscope images like originally desired)
- “PYTORCH Documentation.” *PyTorch Documentation - PyTorch 2.5 Documentation*, pytorch.org/docs/stable/index.html. Accessed 30 Oct. 2024.

References

- I. Askari, Syed Sajjad. “Alex-Net Explanation and Implementation in Tensorflow and Keras.” *Medium*, Medium, 14 Jan. 2023, medium.com/@syedsajjad62/alex-net-explanation-and-implementation-in-tensorflow-and-keras-8047efeb7a0f.
- II. Sanderson, Grant. “But What is a Convolution?” *Youtube*, 3blue1brown, 29 Dec. 2023, https://www.youtube.com/shorts/kpG7I2MOcnI.
- III. Ave Coders. “101 Concepts of Data Science and Machine Learning” *Youtube*, AveCoders, 28 May. 2024, https://www.youtube.com/shorts/ykt5PUxs3z8.
- IV. Chima, Precious. “Activation Functions: ReLU & Softmax.” *Medium*, Medium, 5 Apr. 2020, https://medium.com/@preshchima/activation-functions-relu-softmax-87145bf39288.
- V. Alexander, Giffah. “Softmax Activation function explained” *Youtube*, Giffah_Alexander, 22 Oct. 2024, https://www.youtube.com/shorts/SrJN_hpiuAs.
- VI. Sigmoid function. “Sigmoid function.” *Wikipedia*, Wikipedia, 14 Nov. 2024, https://en.wikipedia.org/wiki/Sigmoid_function.
- VII. Krizhevsky, Sutskever, & Hinton. “ImageNet classification with deep convolutional neural networks.” *Commun. ACM 60*. 2017, accessed 29 October 2024, <https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.

References

- VIII. Kingma, Diederik P. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014), accessed 29 October 2024, <https://arxiv.org/pdf/1412.6980>.
- IX. Gabbard & Miller. “Machine Learning from Scratch: Stochastic Gradient Descent and Adam Optimizer.” *18.0851 Project*, accessed 17 November 2024, <https://www.mit.edu/~jgabbard/assets/18085_Project_final.pdf >.
- X. Bangar, Siddhesh. “VGG-Net Architecture Explained.” *Medium*, Medium, 28 Jun. 2022, https://medium.com/@siddheshb008/vgg-net-architecture-explained-71179310050f.
- XI. He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, accessed 29 October 2024, <https://arxiv.org/pdf/1512.03385v1>.
- XII. Ioffe, Sergey. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *arXiv preprint arXiv:1502.03167* (2015). <https://arxiv.org/pdf/1502.03167>.
- XIII. Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014). <https://arxiv.org/pdf/1409.1556>.
- XIV. Tan, Mingxing, and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." *International conference on machine learning*. PMLR, 2019, accessed 29 October 2024, <https://arxiv.org/pdf/1905.11946>.

References

- XV. Hien, Ngo Le Huy. “Grid Search space Illustration” *ResearchGate*, ResearchGate, https://www.researchgate.net/figure/Grid-Search-space-illustration_fig9_346571789.
- XVI. Sandler, Mark, et al. "Mobilenetv2: Inverted residuals and linear bottlenecks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, <https://arxiv.org/pdf/1801.04381>.
- XVII. Neural Computing and Applications. “Graph for ReLU6 activation function” *ResearchGate*, ResearchGate, https://www.researchgate.net/figure/Grid-Search-space-illustration_fig9_346571789.
- XVIII. Derevianko, Ivan. “BLAZOR - IT’S TIME TO FORGET JAVASCRIPT” *Ivan Derevianko*, Ivan Derevianko, 16 July 2019, https://ivanderevianko.com/2019/07/blazor-its-time-to-forget-javascript.
- XIX. Szegedy, Christian, et al. "Rethinking the inception architecture for computer vision." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, <https://arxiv.org/pdf/1512.00567>.
- XX. “Ultralytics.” *Ultralytics YOLO Docs*, https://docs.ultralytics.com/. Accessed 30 Oct. 2024.
- XXI. Pandey, Abhishek Kumar. “SiLU (Sigmoid Linear Unit) activation function.” *Medium*, Medium, 6 Apr. 2023, https://medium.com/@akp83540/silu-sigmoid-linear-unit-activation-function-d9b6845f0c81.
- XXII. Hidayatullah, Priyanto. “YOLOv8 Architecture Detailed Explanation- A Complete Breakdown” *Youtube*, Dr. Priyanto Hidayatullah, 28 Oct. 2023, https://www.youtube.com/watch?v=HQXhDO7COj8&t=74s&ab_channel=Dr.PriyantoHidayatullah.

References

- XXIII. Terven, Juan, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González. "A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas." *Machine Learning and Knowledge Extraction* 5.4 (2023): 1680-1716, accessed 29 October 2024, <https://arxiv.org/pdf/2304.00501v6>.
- XXIV. Loshchilov, I. "Decoupled weight decay regularization." *arXiv preprint arXiv:1711.05101* (2017), accessed 29 October 2024, <https://arxiv.org/pdf/1711.05101>.
- XXV. Pytorch. “Vision/Torchvision/Models/Alexnet.Py at Main · Pytorch/Vision.” *GitHub*, github.com/pytorch/vision/blob/main/torchvision/models/alexnet.py. Accessed 20 Nov. 2024.
- XXVI. Pytorch. “Vision/Torchvision/Models/ResNet.Py at Main · Pytorch/Vision.” *GitHub*, https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py. Accessed 20 Nov. 2024.
- XXVII. Pytorch. “Vision/Torchvision/Models/VGG.Py at Main · Pytorch/Vision.” *GitHub*, https://github.com/pytorch/vision/blob/main/torchvision/models/vgg.py. Accessed 20 Nov. 2024.
- XXVIII. Nvidia. “DeepLearningExamples/Pytorch/Classification/Convnets/Efficientnet at Master · Nvidia/Deeplearningexamples.” *GitHub*, github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Classification/ConvNets/efficientnet. Accessed 20 Nov. 2024.
- XXIX. Pytorch. “Vision/Torchvision/Models/Inception.Py at Main · Pytorch/Vision.” *GitHub*, https://github.com/pytorch/vision/blob/main/torchvision/models/inception.py. Accessed 20 Nov. 2024.