



CNN - A CLOSER ANALYSIS OF THE EFFICIENTNET MODEL

IMAGE: OBJECT CLASSIFICATION

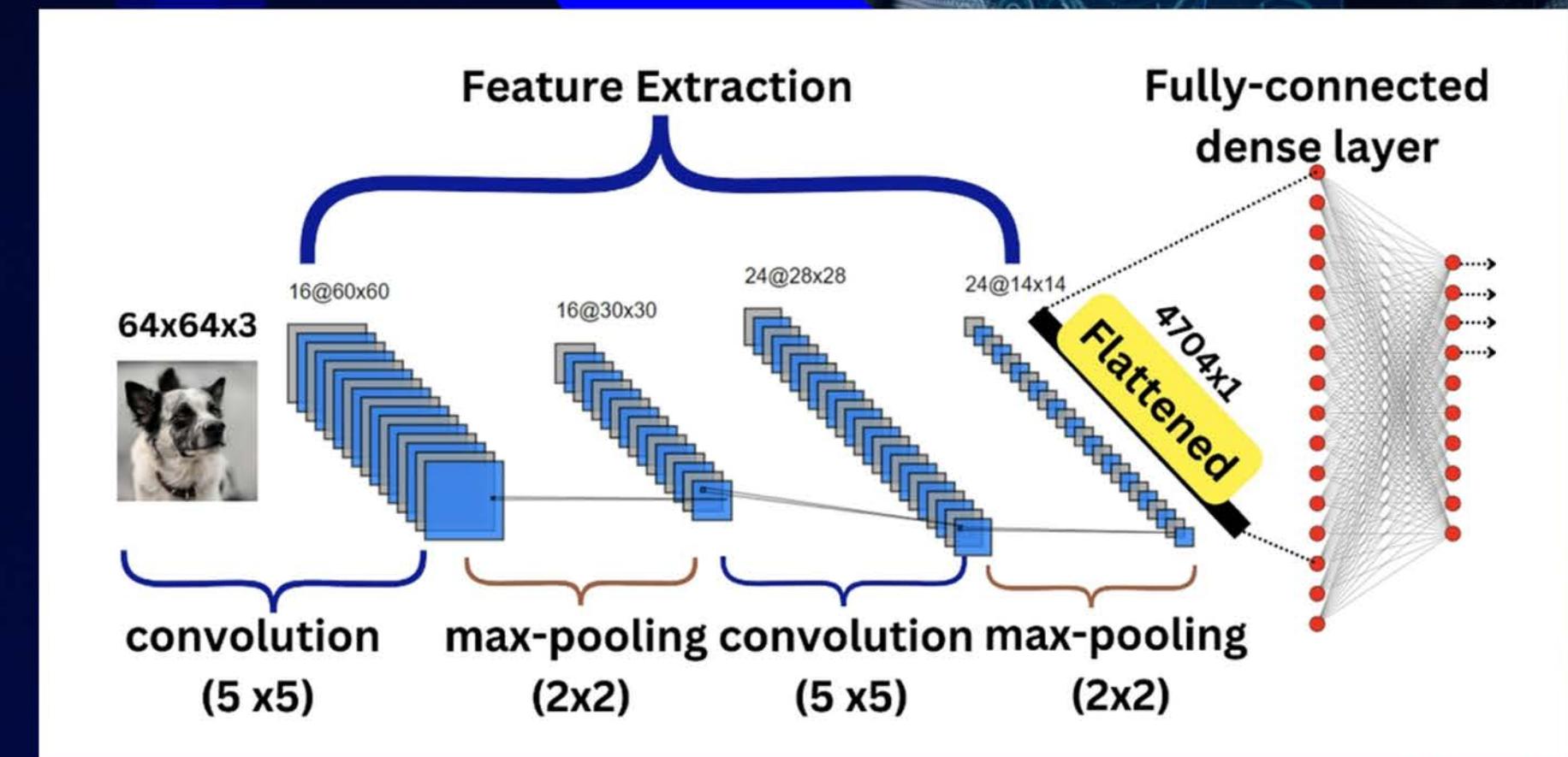
KHIN CHAW | KHIN HPONE



INTRODUCTION

What is a Convolutional Neural Network (CNN)?

- A deep learning model specifically designed for processing grid-like data such as images.
- CNNs automatically learn spatial hierarchies of features, making them ideal for tasks like image classification, object detection, and segmentation.



Example of CNN



WHAT IS EFFICIENTNET?

- EfficientNet is a family of convolutional neural networks developed by Google AI (Tan & Le, 2019).
- It introduces a novel compound scaling method that simultaneously scales:
 - Depth (number of layers)
 - Width (number of channels)
 - Input resolution
- Aimed at achieving high accuracy with low computational cost.
- The EfficientNet family includes models from B0 to B7, where:
 - B0 is the baseline network,
 - B1-B7 progressively scale up in size and performance.
- <https://www.geeksforgeeks.org/computer-vision/efficientnet-architecture/>





WHY EFFICIENTNET IS EFFICIENT

- Uses MBConv (Mobile Inverted Bottleneck Convolution) with:
- Depthwise separable convolution
- Squeeze-and-Excitation (SE) blocks
- Swish activation (instead of ReLU)
- Compound scaling formula:

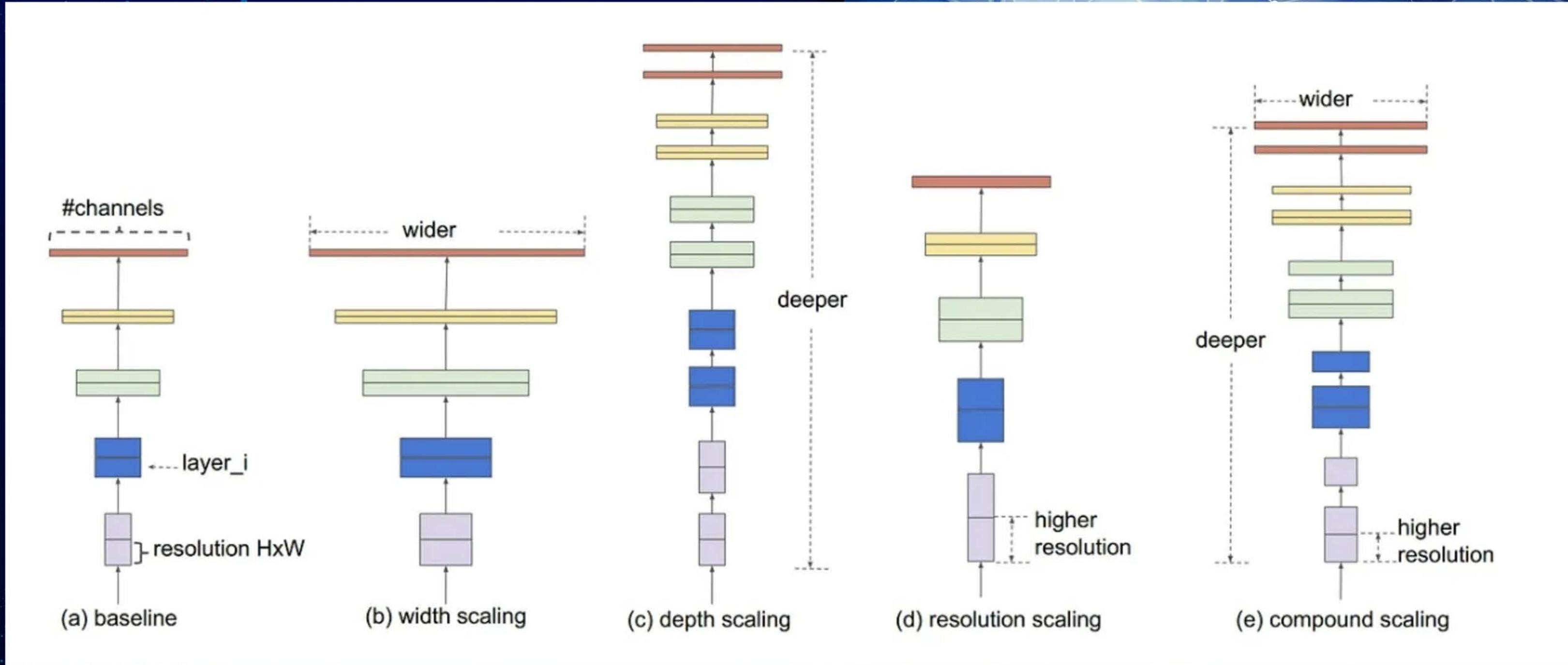
$$\text{depth} = \alpha^\phi, \text{ width} = \beta^\phi, \text{ resolution} = \gamma^\phi, \text{ s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

Width \times Depth² \times Resolution² \approx Constant

EfficientNet-B0 is the baseline; B1–B7 are scaled up versions



EFFICIENTNET



MODEL SCALING REF:Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In Proceedings of the 36th International Conference on Machine Learning, ICML, pages 6105–6114., 2019.

EfficientNet-BO Layer Structure

Stage	Operator	#Channels	#Layers	Resolution	Stride
1	Conv3x3	32	1	224×224	2
2	MBConv1, 3x3	16	1	112×112	1
3	MBConv6, 3x3	24	2	112×112	2
4	MBConv6, 5x5	40	2	56×56	2
5	MBConv6, 3x3	80	3	28×28	2
6	MBConv6, 5x5	112	3	14×14	1
7	MBConv6, 5x5	192	4	14×14	2
8	MBConv6, 3x3	320	1	7×7	1
9	Conv1x1 + Pool	1280	1	7×7	-

- ALL CONVOLUTIONAL LAYERS ARE FOLLOWED BY:
 - BATCH NORMALIZATION
 - SWISH ACTIVATION (BETTER THAN RELU)
- MBCONV INCLUDES SE BLOCK (RECALIBRATION OF CHANNELS)



EFFICIENTNET Layer Components

- ◆ Input Layer
 - Typically uses $224 \times 224 \times 3$ RGB images
 - Applied normalization and standard resizing
- ◆ Stem Layer
 - 3×3 Conv, stride 2, output shape \downarrow from 224×224 to 112×112
 - Uses swish activation function ($f(x) = x * \text{sigmoid}(x)$)
- ◆ MBConv Blocks (Mobile Inverted Bottleneck Convolution)
 - Each block includes:
 - 1×1 Pointwise conv (expansion)
 - 3×3 or 5×5 Depthwise conv
 - 1×1 Pointwise conv (projection)
 - Squeeze-and-Excitation (SE) block (channel-wise recalibration)
 - Replaces traditional convolutions blocks for better efficiency
- ◆ Final Layers
 - 1×1 Conv \rightarrow Global Average Pooling \rightarrow Dropout \rightarrow Fully Connected Layer (e.g., 1000-class for ImageNet)



Compound Scaling

EfficientNet scales:

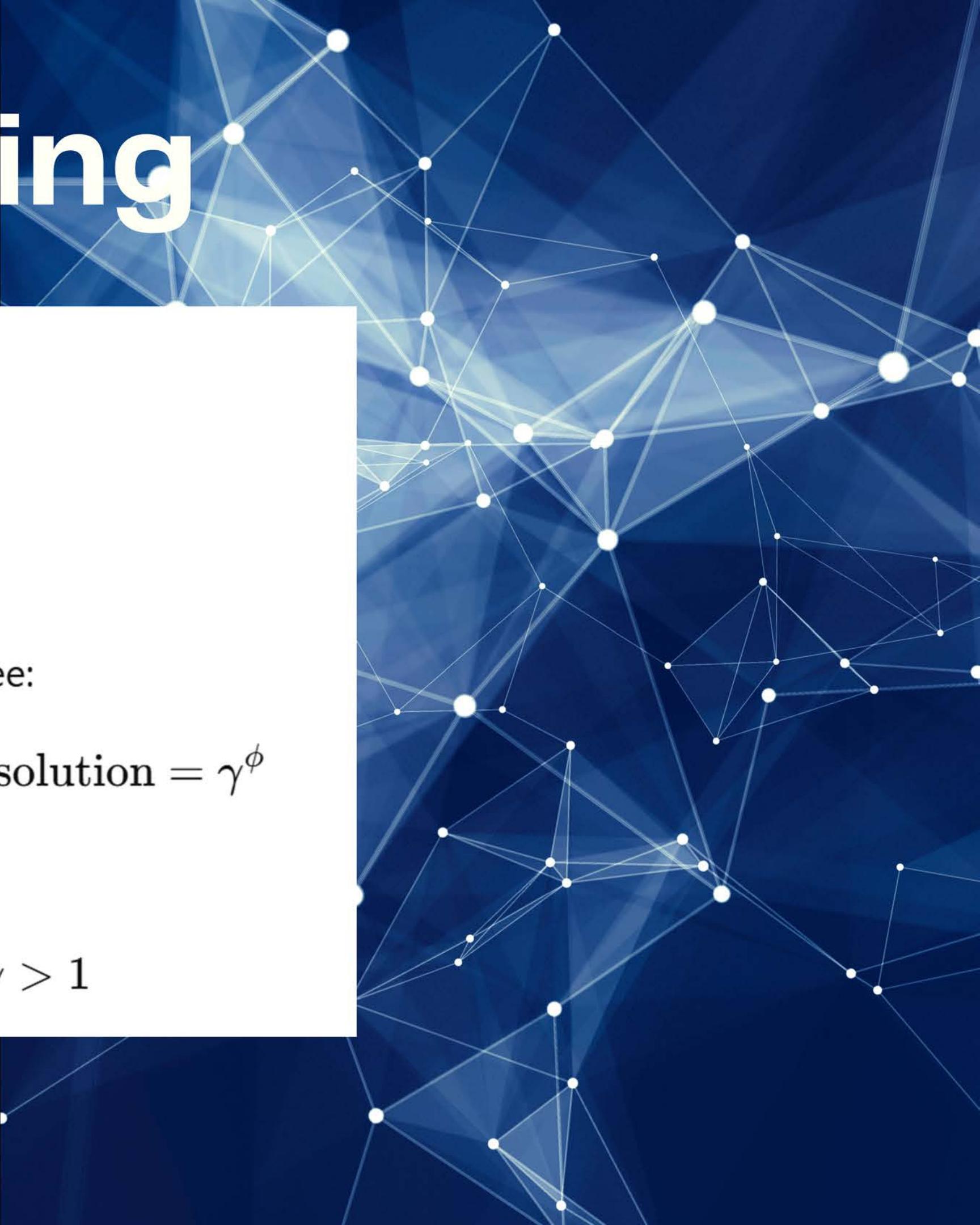
- **Width (w)** → # of channels
- **Depth (d)** → # of layers
- **Resolution (r)** → input image size

Using a **compound coefficient φ** , it balances all three:

$$\text{depth} = \alpha^\varphi, \quad \text{width} = \beta^\varphi, \quad \text{resolution} = \gamma^\varphi$$

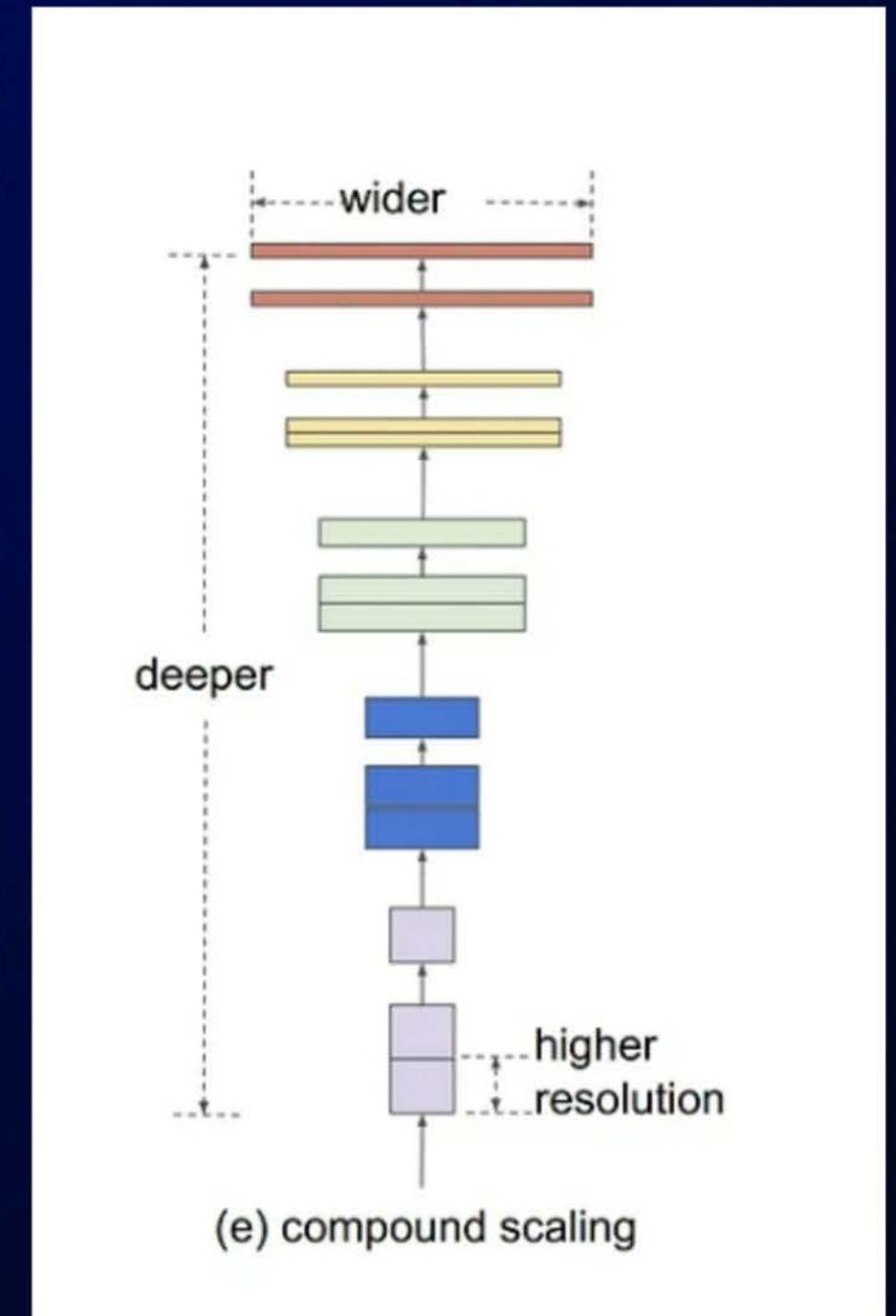
Subject to constraint:

$$\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2, \quad \alpha, \beta, \gamma > 1$$



Compound Scaling in Action

- **Width scaling:** Adds more channels per layers
- **Depth scaling:** Adds more layers
- **Resolution scaling:** Increases input image size
- **Compound scaling is more balanced and effective than scaling one factor alone.**



[“\[https://www.tensorflow.org/api_docs/python/tf/keras/applications/EfficientNetB0\]\(https://www.tensorflow.org/api_docs/python/tf/keras/applications/EfficientNetB0\)](https://www.tensorflow.org/api_docs/python/tf/keras/applications/EfficientNetB0)

Kernel Sizes & Configurations

Block Type	Kernel Size	Stride	Expansion Ratio	Repeats
MBConv1	3×3	1	1	1
MBConv6	3×3 or 5×5	2	6	varies

- Varies across EfficientNet-B0 to B7.
- Each version (B0 to B7) increases model capacity by scaling depth, width, and resolution using compound scaling
- Depth-wise separable convolutions reduce FLOPs (Floating Point Operations), reduces model size & parameters, maintains high accuracy.

TensorFlow v2.16.1

TensorFlow > API > TensorFlow v2.16.1 > Python

Was this helpful?

tf.keras.applications.EfficientNetB0

On this page

Used in the notebooks

Args

Returns

[View source on GitHub](#)

Instantiates the EfficientNetB0 architecture.

[View aliases](#)

```
tf.keras.applications.EfficientNetB0(  
    include_top=True,  
    weights='imagenet',  
    input_tensor=None,  
    input_shape=None,  
    pooling=None,  
    classes=1000,  
    classifier_activation='softmax',  
    **kwargs
```

[“https://www.tensorflow.org/api_docs/python/tf/keras/applications/EfficientNetB0”](https://www.tensorflow.org/api_docs/python/tf/keras/applications/EfficientNetB0)

LIMITATIONS

- Higher inference time on CPU-only environments
- Difficult to customize compared to plain CNNs due to compound scaling
- SE blocks and Swish activation are computationally expensive, especially on edge/mobile devices

Use Cases & Applications

- Best suited for high-accuracy tasks where sufficient resources are available
- Applied in: medical imaging (e.g., tumor detection), retail product detection, mobile apps (EffNet-Lite)



Project Overview

Comparative Study of CNN Architectures for Image Classification : Focusing on EfficientNet

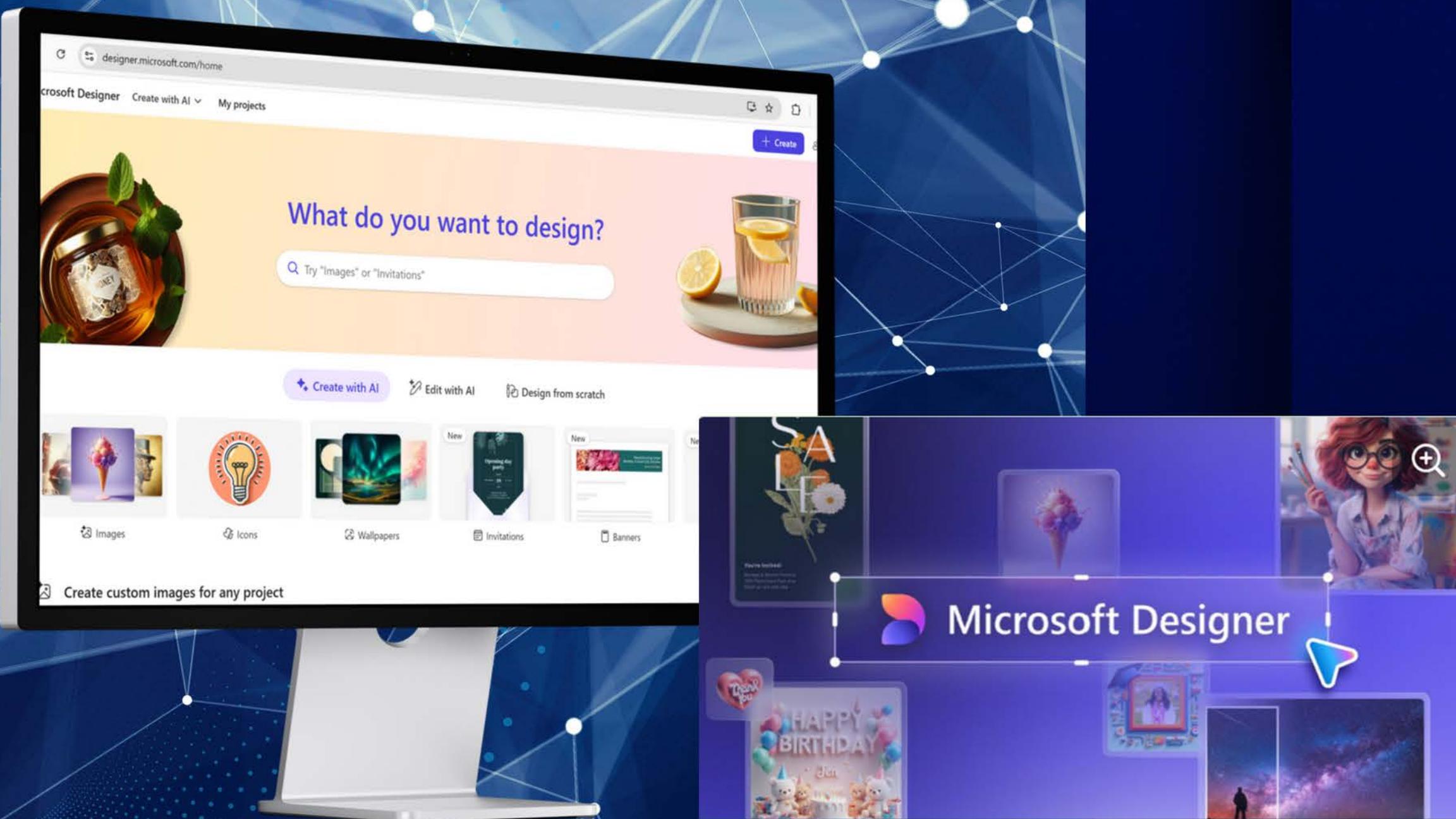
- explored the performance of five popular Convolutional Neural Network (CNN) architectures using Python and Keras.
- The models used are: ResNet50, VGG16, InceptionV3, ConvNeXt-Tiny, and EfficientNetB7.
- used a dataset of 40 images (20 real and 20 synthetic) to test the EfficientNet model.

GOAL

TO COMPARE EACH MODEL'S PERFORMANCE IN TERMS OF PREDICTION ACCURACY, INFERENCE SPEED, AND PRACTICAL USABILITY.



MICROSOFT IMAGE GENERATOR



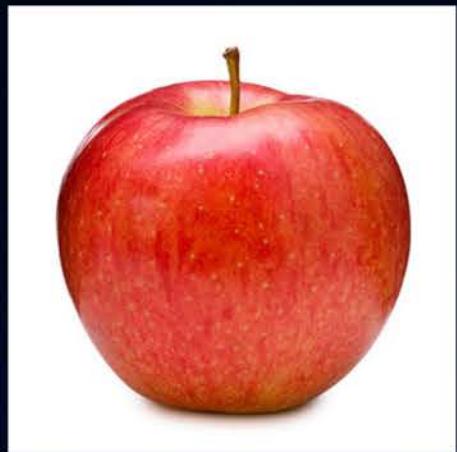
Microsoft. Microsoft image creator.

"<https://designer.microsoft.com/image-creator>", 2025.

DATASET

- The dataset contains:
- 20 real images collected from personal captured photo (and internet)
- 20 synthetic images generated by Microsoft's AI image generator
- Real images vary in size; synthetic images are fixed at 1024×1024 pixels.
- All images were preprocessed and resized to fit model's input shape .

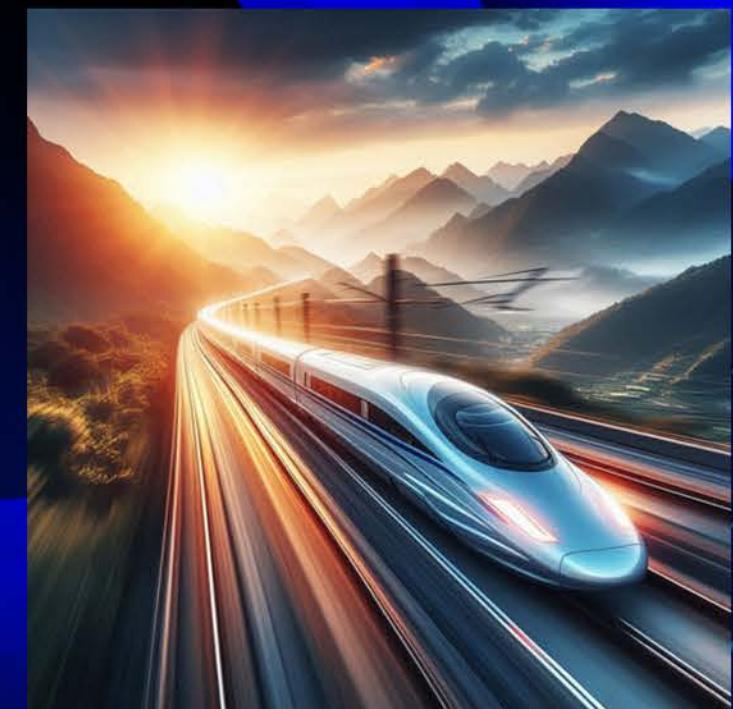
DATASET-REAL AND SYNTHETIC



DATASET-REAL AND SYNTHETIC



DATASET-REAL AND SYNTHETIC



EXPERIMENT

The screenshot shows a Jupyter Notebook interface with several windows open:

- File Explorer:** Shows the project structure under "FOLDERS: CHAPTER3".
- Terminal:** Displays Python commands for mounting Google Drive and listing files in a directory.
- Code Editor:** The main window displays the code for `cnn_models.py`. The code defines a class `cnnModels` with methods for different pre-trained models: `resnet`, `vggnet`, `inception`, `convnet`, and `efficientnet`. Each method returns a corresponding Keras model. The `__init__` method initializes the class with a dictionary of models. A helper function `get_predictions` is also defined to process image files.
- Output:** Shows the results of running the code, including a DataFrame `result_df` and a terminal log indicating data download from a GitHub repository.

- All models were pre-trained on the ImageNet dataset using Keras.

IMAGE CLASSIFICATION

```
6
7     for model in ['ResNet50', 'VGGNet16', 'InceptionV3', 'ConvNeXt', 'EfficientNet']
8     :
9         table_df[model] = df.apply(lambda row: format_top1(row[model], row[f'{model}_prob']), axis=1)
10
11 table_df.head()
```

filename	ResNet50	VGGNet16	InceptionV3	ConvNeXt	EfficientNet
0 cow	muzzle (0.39)	miniature_pinscher (0.08)	hammer (0.99)	hen (0.07)	chainlink_fence (0.12)
1 toy	ballpoint (0.22)	pencil_box (0.25)	pitcher (0.58)	tennis_ball (0.39)	saltshaker (0.07)
2 flower	pot (0.26)	sulphur_butterfly (0.19)	flatworm (0.67)	hummingbird (0.39)	reel (0.11)
3 dog	Persian_cat (0.83)	Persian_cat (0.67)	web_site (1.00)	Persian_cat (0.59)	Arctic_fox (0.44)
4 bird	African_grey (0.95)	quail (0.48)	web_site (0.99)	African_grey (0.86)	window_screen (0.50)

```
1 import pandas as pd
2
3 df = pd.read_csv('/content/gdrive/MyDrive/DeepLearning/Project_01/results/
4 fake_result.csv')
5 df.head() # or df[['label', 'ResNet50', 'ResNet50_prob', ...]]
```

	ResNet50	VGGNet16	InceptionV3	ConvNeXt	EfficientNet	ResNet50_prob	VGGNet16_prob	InceptionV3_p
0 Pembroke	sunglasses	stopwatch	fountain	sunglass	0.415618	0.422213	0.999	
1 hen	cock	clog	hen	hen	0.539845	0.519462	0.999	
2 car_mirror	mountain_tent	clog	bullet_train	bullet_train	0.556300	0.316349	0.916	
3 hamper	Persian_cat	flatworm	tabby	tiger_cat	0.361622	0.572489	0.999	
4 candle	candle	stole	bakery	stove	0.674406	0.387122	0.996	

```
1 import pandas as pd
2
3 df = pd.read_csv('/content/gdrive/MyDrive/DeepLearning/Project_01/results/
4 real_result.csv')
5 df.head() # or df[['label', 'ResNet50', 'ResNet50_prob', ...]]
```

	ResNet50	VGGNet16	InceptionV3	ConvNeXt	EfficientNet	ResNet50_prob	VGGNet16_prob	Incep
0	muzzle	miniature_pinscher	hammer	hen	chainlink_fence	0.387831	0.080818	
1	ballpoint	pencil_box	pitcher	tennis_ball	saltshaker	0.219144	0.247788	
2	pot	sulphur_butterfly	flatworm	hummingbird	reel	0.256225	0.187836	
3	Persian_cat	Persian_cat	web_site	Persian_cat	Arctic_fox	0.832892	0.668904	
4	African_grey	quail	web_site	African_grey	window_screen	0.945016	0.481908	

IMAGE CLASSIFICATION: REAL

	A	B	C	D	E	F	G	H	I	J	K
1	ResNet50	VGGNet16	InceptionV3	ConvNeXt	EfficientNet	ResNet50_prot	VGGNet16_pro	InceptionV3_pr	ConvNeXt_prot	EfficientNet_pr	label
2	muzzle	miniature_pins	hammer	hen	chainlink_fence	0.3878315	0.08081827	0.99297863	0.07476012	0.11729001	cow
3	ballpoint	pencil_box	pitcher	tennis_ball	saltshaker	0.21914423	0.2477882	0.58344233	0.38547233	0.07368472	toy
4	pot	sulphur_butterfly	flatworm	hummingbird	reel	0.256225	0.18783566	0.6741588	0.3888667	0.10678629	flower
5	Persian_cat	Persian_cat	web_site	Persian_cat	Arctic_fox	0.8328918	0.66890377		1	0.59154844	0.43703032
6	African_grey	quail	web_site	African_grey	window_screen	0.94501597	0.4819076	0.9906729	0.85805225	0.5037069	bird
7	plate	plate	pitcher	wok	strainer	0.67530185	0.65339124	0.99392855	0.3796683	0.77589095	cake
8	passenger_car	bullet_train	sock	bullet_train	cab	0.90250206	0.9173202	0.81538737	0.45826787	0.2625382	train
9	canoe	canoe	hammer	canoe	canoe	0.3432207	0.4159075	0.99999905	0.2534945	0.69345	boat
10	chimpanzee	chimpanzee	ashcan	chimpanzee	chimpanzee	0.99999964	0.9903126		1	0.863563	0.80125403
11	diaper	diaper	web_site	diaper	diaper	0.92203164	0.81497616	0.9955249	0.40672663	0.60209274	baby
12	tiger	tiger	pitcher	tiger	tiger_cat	0.86475635	0.84903365	0.9999975	0.7405015	0.5470659	tiger
13	Granny_Smith	pomegranate	web_site	Granny_Smith	Granny_Smith	0.29145214	0.5377776	0.9841814	0.92453253	0.24539247	apple
14	rubber_eraser	rubber_eraser	clog	rubber_eraser	rubber_eraser	0.85895944	0.6875064	0.9994199	0.8447058	0.23501019	pencil
15	canoe	lakeside	web_site	lakeside	alp	0.6804078	0.4947493	0.9955877	0.4838826	0.12623677	lake
16	dining_table	dining_table	stopwatch	dining_table	window_screen	0.92214185	0.73553246	0.9999585	0.9642549	0.22803657	dining_table
17	school_bus	passenger_car	binoculars	passenger_car	limousine	0.9107605	0.3958574	0.47202057	0.78599834	0.07698787	school_bus
18	shopping_basket	shopping_basket	pitcher	shopping_basket	shopping_basket	0.999997	0.9999981	0.5271109	0.90707844	0.7847785	plastic_basket
19	hen	hen	flatworm	hen	window_screen	0.91908205	0.8252454		1	0.91907793	0.09330011
20	water_bottle	water_bottle	flatworm	cocktail_shaker	water_bottle	0.99084634	0.9098153	0.9999862	0.6170162	0.7646816	water_bottle
21	tabby	tabby	web_site	Egyptian_cat	tabby	0.5086882	0.41537073		1	0.224414	0.533094
22											



IMAGE CLASSIFICATION: FAKE

	A	B	C	D	E	F	G	H	I	J	K
1	ResNet50	VGGNet16	InceptionV3	ConvNeXt	EfficientNet	ResNet50_prot	VGGNet16_pro	InceptionV3_pr	ConvNeXt_prot	EfficientNet_pr	label
2	Pembroke	sunglasses	stopwatch	fountain	sunglass	0.41561753	0.42221278	0.99936837	0.27203736	0.07401339	cow
3	hen	cock	clog	hen	hen	0.5398454	0.5194624	0.99939394	0.7929004	0.07392945	hen
4	car_mirror	mountain_tent	clog	bullet_train	bullet_train	0.55630046	0.3163486	0.91653734	0.54976016	0.72447175	train
5	hamper	Persian_cat	flatworm	tabby	tiger_cat	0.3616216	0.572489	0.9999889	0.23659141	0.46088853	cat
6	candle	candle	stole	bakery	stove	0.6744064	0.38712204	0.99627614	0.25372165	0.10477709	cake
7	toyshop	toyshop	clog	toyshop	teddy	0.99368423	0.93309903	0.9999989	0.8793247	0.4359823	toy
8	tiger	fountain	flatworm	tiger	tiger_cat	0.5448647	0.13387209	0.9999994	0.85821325	0.27814662	tiger
9	school_bus	school_bus	flatworm	school_bus	church	0.9273219	0.9992399	0.9999975	0.8030589	0.6982585	school_bus
10	water_bottle	water_bottle	stopwatch	water_bottle	water_bottle	0.59653836	0.26066345	0.9985195	0.91486984	0.8406696	water_bottle
11	yawl	yawl	stopwatch	yawl	catamaran	0.7840371	0.34552598	0.9967843	0.77020305	0.2344706	boat
12	indigo_bunting	hummingbird	flatworm	indigo_bunting	jay	0.37666985	0.25452876	0.99821943	0.4781175	0.56312877	bird
13	pinwheel	umbrella	clog	umbrella	matchstick	0.9226753	0.7096159	1	0.34280324	0.70447534	pencil
14	comic_book	mask	flatworm	comic_book	comic_book	0.6911051	0.45315897	0.8420374	0.25946134	0.14329687	monkey
15	shopping_bask	shopping_bask	flatworm	shopping_bask	shopping_bask	0.9543658	0.47739467	0.83421165	0.6977798	0.6070933	plastic_basket
16	boathouse	volcano	web_site	mashed_potato	web_site	0.5870327	0.28246665	1	0.62775177	0.030386461	lake
17	bee	daisy	flatworm	daisy	matchstick	0.3811552	0.3241558	0.76654124	0.25772262	0.052193206	flower
18	golden_retrieve	golden_retrieve	flatworm	golden_retrieve	golden_retrieve	0.97924495	0.84636927	0.9997991	0.8799979	0.5961258	dog
19	restaurant	restaurant	flatworm	dining_table	window_screen	0.7806092	0.8331283	0.9968098	0.6387252	0.1026649	dining_table
20	Granny_Smith	lipstick	web_site	Granny_Smith	strainer	0.48318785	0.21608412	1	0.87527907	0.6327593	apple
21	maraca	candle	flatworm	wig	pill_bottle	0.442965	0.801257	0.99996054	0.48872933	0.21910615	baby
22											



MODEL ACCURACY

ResNet50: Real = 25%, Fake = 25%

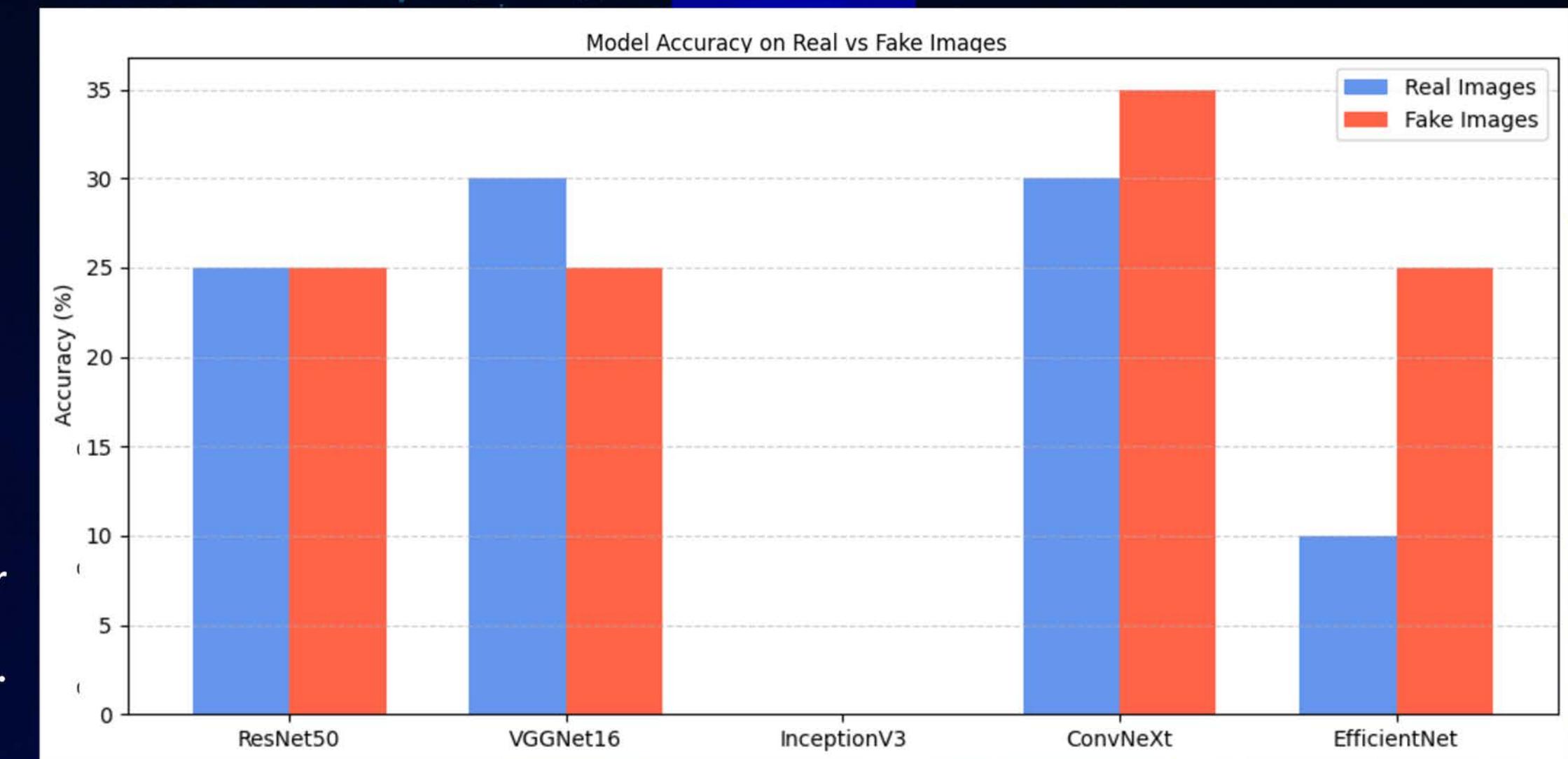
VGGNet16: Real = 30%, Fake = 25%

InceptionV3: Real = 0%, Fake = 0%

ConvNeXt: Real = 30%, Fake = 35%

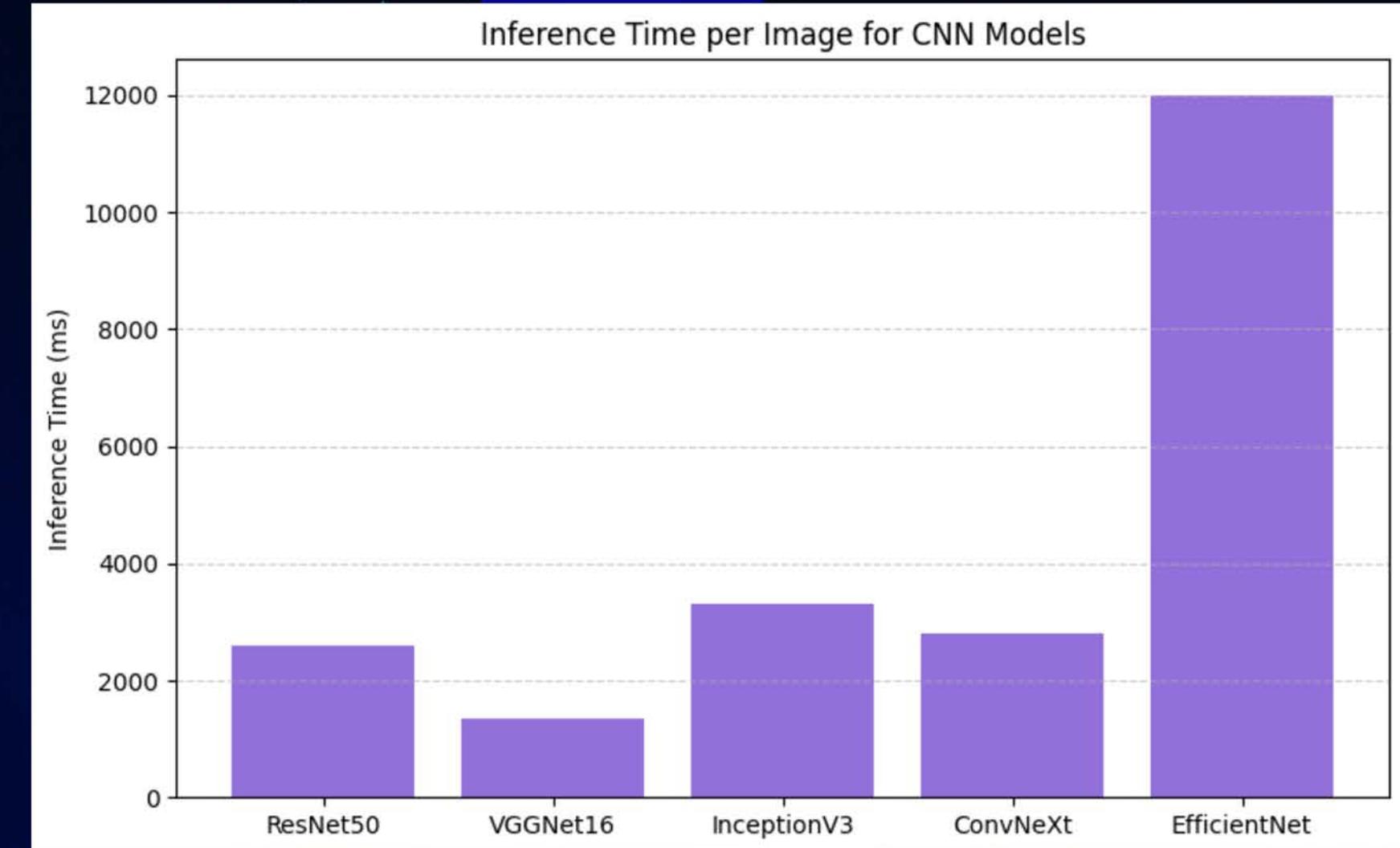
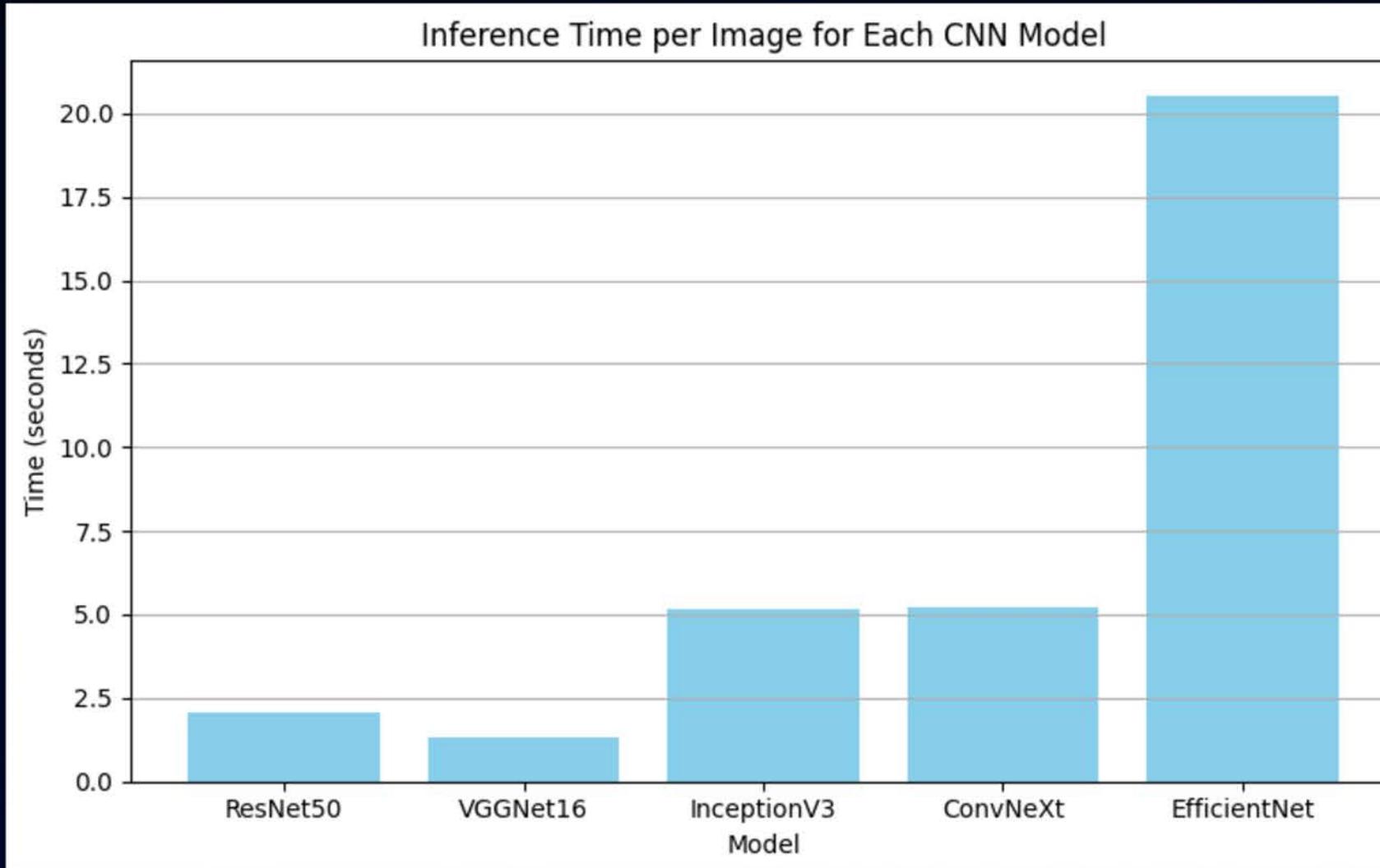
EfficientNet: Real = 10%, Fake = 25%

Observation: ConvNeXt showed the highest accuracy for fake images (35%) and tied with VGGNet16 on real (30%).
InceptionV3 failed to recognize both.



- ConvText-Tiny is close with good trade-off between accuracy and speed.

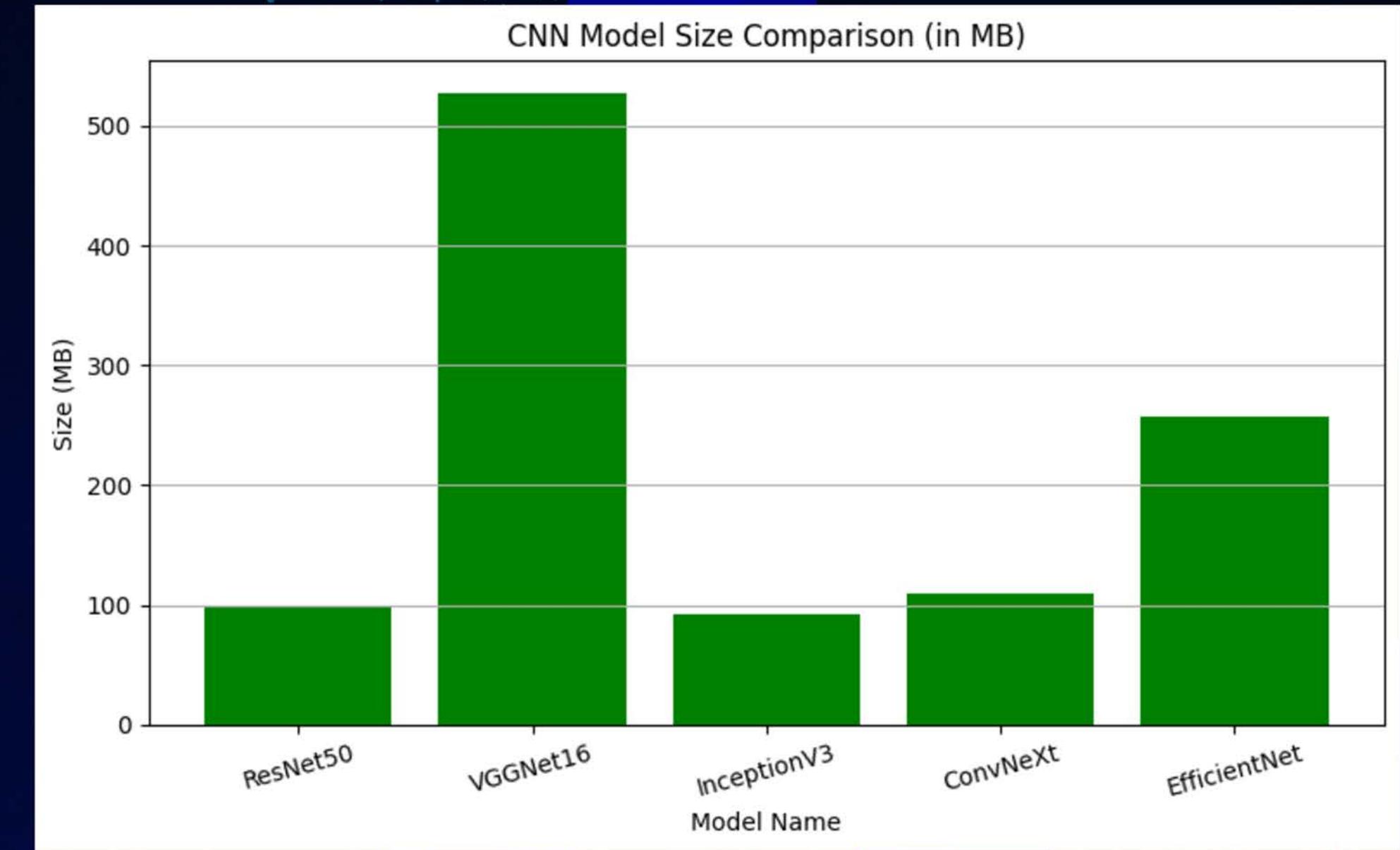
Inference Time per Image



- Fastest: **VGGNet16** (~1s)
- Slowest: **EfficientNet** (~11s)
- Observation: EfficientNet, while being accurate in some cases, had the longest inference time.

Model size and Memory footprint

Model	File Size (MB)
ResNet50	98.35
VGG16	527.87
InceptionV3	91.93
ConvNeXt-Tiny	109.53
EfficientNetB7	256.59



- Smallest: ResNet50 (~98 MB)
- Largest: VGGNet16 (~528 MB)
- Observation: VGGNet16 is over 5x the size of ResNet50 with only slightly better accuracy.

FINDINGS

- Best Overall Tradeoff: ConvNeXt – highest accuracy (real: 30%, fake: 35%) and moderate inference time and size
- Best for Speed: VGGNet16 – fastest inference, but very large size
- Worst Performer: InceptionV3 – 0% accuracy despite moderate speed and size
- Insight: Model size doesn't directly correlate with accuracy or speed



LIMITATIONS

- Small dataset: Limited number of images used for evaluation
- No fine-tuning: All models used pre-trained ImageNet weights without custom training
- No qualitative error analysis: Misclassified images not visually inspected

FUTURE WORK

- Expand dataset size and diversity for more reliable benchmarking
- Fine-tune models on a custom dataset to increase domain-specific performance
- Test on different image types (e.g., grayscale, occluded, rotated) for robustness





CNN - A CLOSER ANALYSIS OF THE EFFICIENTNET MODEL

IMAGE: OBJECT CLASSIFICATION

KHIN CHAW | KHIN HPONE

