

# Deep Fake Image Detection Using Deep CNN Architecture

By: Devin Liu & Khiem Do

>>>>

## Introduction

.....

### What are deepfakes?

- Realistic but fake synthetic media generated by AI.
- Created using Generative Adversarial Networks.
- Used for entertainment purposes but pose risks.

### Why detect deepfakes?

- Distinguishing between real and fake content becomes increasingly challenging.
- Essential to combat misinformation and ensure media authenticity in today's digital world.

### Objective of the Project

- Develop an improved CNN model for deepfake detection.
- Focus on achieving high accuracy while maintaining computational efficiency.

## Problem Statement

>>>>

.....

### Deepfakes' Challenges

- Realism
- Misinformation
- Detection Complexity

### Current Limitations

- Lack of Generalizability
- Robustness Issues
- Overfitting

### Solution

- Improved Dense CNN Architecture
- Focus on Robustness

## Overview of Proposed Model

>>>>

### Based on Dense CNN Architecture

### Key Improvements

### Binary Classification Task

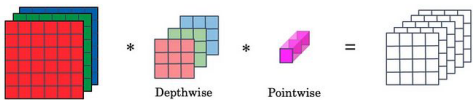
### Expected Benefits

# CNN vs MobileNet

## Normal Convolution



## Depthwise Separable Convolution



Depthwise

Pointwise

## Comparing layers

Layer	Layer Type	Output Dimension	No. of Parameters
1	Input (Convolution 2-D)	(160 x 160 x 3)	255
2	Batch Normalization	(160 x 160 x 3)	12
3	Convolution 2-D	(160 x 160 x 16)	1168
4	Convolution 2-D	(160 x 160 x 16)	7320
5	Batch Normalization	(160 x 160 x 16)	64
6	Average Pooling 2-D	(80 x 80 x 16)	0
7	Convolution 2-D	(80 x 80 x 32)	4640
8	Convolution 2-D	(80 x 80 x 32)	9248
9	Convolution 2-D	(80 x 80 x 32)	9248
10	Batch Normalization	(80 x 80 x 32)	128
11	Average Pooling 2-D	(40 x 40 x 32)	0
12	Convolution 2-D	(40 x 40 x 64)	18496
13	Convolution 2-D	(40 x 40 x 64)	96928
14	Convolution 2-D	(40 x 40 x 64)	96928
15	Convolution 2-D	(40 x 40 x 64)	96928
16	Batch Normalization	(40 x 40 x 64)	256
17	Average Pooling 2-D	(20 x 20 x 64)	0
18	Convolution 2-D	(20 x 20 x 128)	204808
19	Batch Normalization	(20 x 20 x 128)	512
20	Max Pooling 2-D	(10 x 10 x 128)	0
21	Convolution 2-D	(10 x 10 x 256)	419456
22	Batch Normalization	(1 x 10 x 256)	1024
23	Max Pooling 2-D	(5 x 5 x 256)	0
24	Flatten	(6,400)	0
25	Dense	(512)	204832
26	Leaky ReLU	(512)	0
27	Dense	(512)	0
28	Leaky ReLU	(512)	0
29	Dense	(128)	0
30	Leaky ReLU	(128)	0
31	Dense	(16)	0
32	Leaky ReLU	(16)	0
33	Dense	(16)	0
34	Leaky ReLU	(16)	0
35	Dense	(1)	0
Total Parameters			1,786,177
Trainable Parameters			1,787,169
Non-Trainable Parameters			1,008

Layer	Layer Type	Output Dimension
1	Input (Dense 48)	(48 x 48 x 8)
2	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
3	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
4	Average Pooling 4D	(6 x 6 x 48)
5	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
6	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
7	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
8	Average Pooling 4D	(6 x 6 x 48)
9	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
10	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
11	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
12	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
13	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
14	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
15	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
16	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
17	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
18	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
19	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
20	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
21	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
22	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
23	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
24	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
25	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
26	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
27	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
28	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
29	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
30	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
31	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
32	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
33	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
34	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
35	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
36	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
37	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
38	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
39	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
40	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
41	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
42	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
43	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
44	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
45	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
46	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
47	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
48	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
49	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
50	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
51	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
52	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
53	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
54	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
55	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
56	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
57	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
58	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
59	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
60	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
61	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
62	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
63	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
64	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
65	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
66	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
67	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
68	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
69	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
70	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
71	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
72	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
73	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
74	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
75	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
76	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
77	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
78	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
79	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
80	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
81	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
82	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
83	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
84	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
85	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
86	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
87	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
88	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
89	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
90	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
91	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
92	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
93	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
94	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
95	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
96	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
97	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
98	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
99	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)
100	MobileBlock (Depthwise + Pointwise)	(48 x 48 x 48)

## Parameter Math

Standard Convolution Layer:

- Parameters = (kernel size<sup>2</sup> x input channels) x output channels

Separable Depthwise Convolution Layer:

Depthwise Convolution:

- Parameters = (kernel size<sup>2</sup>) x input channels

Pointwise Convolution:

- Parameters = input channels x output channels

Total:

- Parameters ≈ Input channels x (kernel size<sup>2</sup> + output channels)

Reduction factor ≈ (1/output channels) + (1/(kernel size<sup>2</sup>))

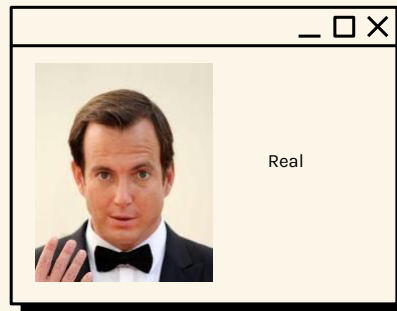
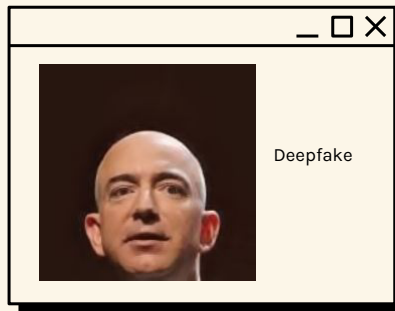
Let N = 512, K = 3, 3

(1/512) + (1/(3<sup>2</sup>)) = 0.113 About 9 times less parameters than standard Convolution

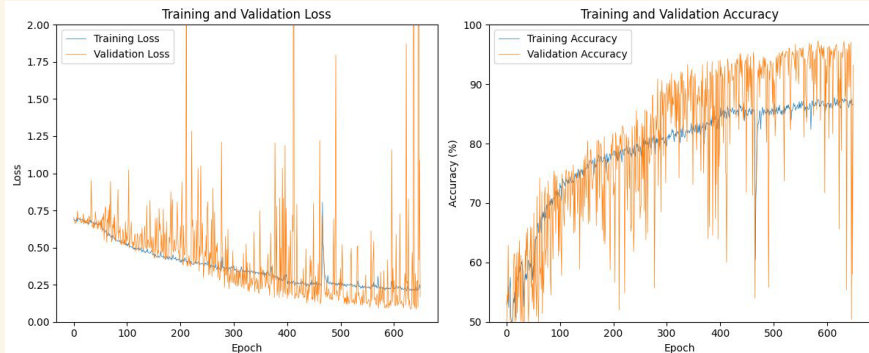
# Data & Augmentation

- **Datasets used:**
  - **Real Images:** CelebA & FFHQ datasets
  - **Deepfake Images:** Generated using various GAN architectures - StyleGAN, StarGAN, AttGAN and GDWCT.
- **Data Splits:**
  - **Training:** 60% of the dataset
  - **Validation:** 10% of the dataset
  - **Testing:** 30% of the dataset
  - Split & balanced to prevent bias and improve performance on unseen data.
- **Data Augmentation Techniques:**
  - **Random Rotations, Flips and Resizing**
  - **Goal:** Prevent overfitting and enhance model's ability to generalize to new, unseen images.

## Data example



## Training and Validation



## Results

### Accuracy

How often model correctly classifies samples

**95.3%**

### Recall

Proportion of actual positive samples correctly predicted

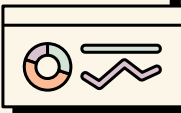
**94.2%**

### Precision

Proportion of correctly predicted positive out of all sample predicted positive

### F1-Score

harmonic mean of precision and recall



# Conclusion

- Achieved 96.83% accuracy with good precision, recall, and f-1 metrics.
- Provide a lightweight solution suitable for real-world deepfake detection.
- To-do: improve generalizability across various datasets, not just limited to GANs.

```
File Edit Selection View Go Run Terminal Help
modelnet_deepfake_detector.py:14 X
+ Code + Notebook | Go to File | Go to Line | Go to Symbol | Clear All Outputs | Variables | Outline | jupyter python 3.10.2
# Uncomment one of these to test image. You can provide your own path as well***
# test_image_path = "manual_test_images/real/fake_1.jpg"
# test_image_path = "manual_test_images/real/fake_2.jpg"
# test_image_path = "manual_test_images/real/fake_3.jpg"
test_image_path = "manual_test_images/real/fake_4.jpg"

test_image = image.open(test_image_path)

test_image = transform(test_image).unsqueeze(0).to(device) # add batch dimension

pretrained_model.to(device)
pretrained_model.eval()

with torch.no_grad():
    output = pretrained_model(test_image)
    probability = output.item()

    if probability < 0.5:
        print("Prediction: Deepfake (Probability: (probability * 100).47%)")
    else:
        print("Prediction: Real (Probability: (probability * 100).47%)")

Python

Precision: 0.9643
Recall: 0.9727
F1 Score: 0.9685

***Uncomment one of these to test image. You can provide your own path as well***
# test_image_path = "manual_test_images/real/fake_1.jpg"
# test_image_path = "manual_test_images/real/fake_2.jpg"
# test_image_path = "manual_test_images/real/fake_3.jpg"
test_image_path = "manual_test_images/real/fake_4.jpg"

test_image = image.open(test_image_path)

test_image = transform(test_image).unsqueeze(0).to(device) # add batch dimension

pretrained_model.to(device)
pretrained_model.eval()

with torch.no_grad():
    output = pretrained_model(test_image)
    probability = output.item()

    if probability < 0.5:
        print("Prediction: Deepfake (Probability: (probability * 100).47%)")
    else:
        print("Prediction: Real (Probability: (probability * 100).47%)")

Python
```

