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 post 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

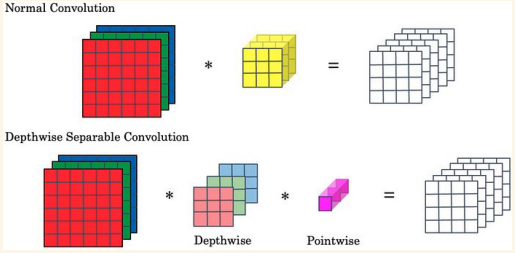
Based on Dense CNN Architecture

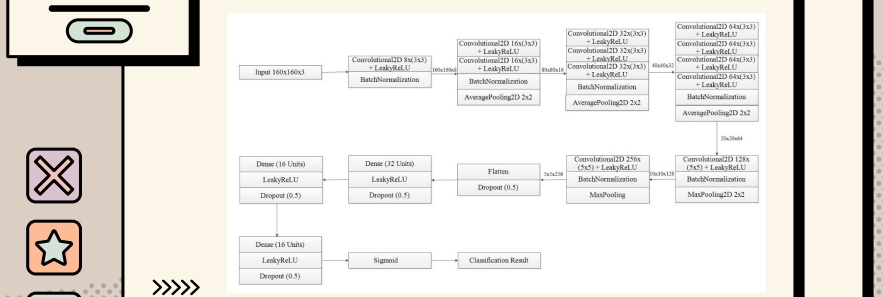
Key Improvements

Binary Classification Task

Expected Benefits

CNN vs MobileNet





Comparing layers

Layer	Layer Type	Output Dimension	No. of Parameters
1	Input (Convolution 2-D)	(160 x 160 x 8)	224
2	Batch Normalization	(160 x 160 x 8)	32
3	Convolution 2-D	(160 x 160 x 16)	1168
4	Convolution 2-D	(160 x 160 x 16)	2320
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)	36928
14	Convolution 2-D	(40 x 40 x 64)	36928
15	Convolution 2-D	(40 x 40 x 64)	36928
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)	204928
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)	819456
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	Dropout	(6,400)	0
26	Dense	(32)	204832
27	Leaky ReLU	(32)	0
28	Dropout	(32)	0
29	Dense	(16)	528
30	Leaky ReLU	(16)	0
31	Dropout	(16)	0
32	Dense	(16)	272
33	Leaky ReLU	(16)	0
34	Dropout	(16)	0
35	Dense	(1)	17
	Total Parameters	1,388,177	
	Trainable Parameters	1,387,169	
	Non-Trainable Parameters	1,008	

Layer	Layer Type	Output Dimension
1	Input (Conv 2D)	(160 x 160 x 8)
2	MobileBlock (Depthwise + Pointwise)	(160 x 160 x 16)
3	MobileBlock (Depthwise + Pointwise)	(160 x 160 x 16)
4	Average Pooling 2D	(80 x 80 x 16)
5	MobileBlock (Depthwise + Pointwise)	(80 x 80 x 32)
6	MobileBlock (Depthwise + Pointwise)	(80 x 80 x 32)
7	MobileBlock (Depthwise + Pointwise)	(80 x 80 x 32)
8	Average Pooling 2D	(40 x 40 x 32)
9	MobileBlock (Depthwise + Pointwise)	(40 x 40 x 64)
10	MobileBlock (Depthwise + Pointwise)	(40 x 40 x 64)
11	MobileBlock (Depthwise + Pointwise)	(40 x 40 x 64)
12	MobileBlock (Depthwise + Pointwise)	(40 x 40 x 64)
13	Average Pooling 2D	(20 x 20 x 64)
14	MobileBlock (Depthwise + Pointwise, kernel size=5)	(20 x 20 x 128)
15	Max Pooling 2D	(10 x 10 x 128)
16	MobileBlock (Depthwise + Pointwise, kernel size=5)	(10 x 10 x 256)
17	Max Pooling 2D	(5 x 5 x 256)
18	Flatten	(1 x 6400)
19	Dropout	(1 x 6400)
20	Fully Connected (FCs)	(1 x 32)
21	Leaky ReLU Activation	(1 x 32)
22	Dropout	(1 x 32)
23	Fully Connected (FCs)	(1 x 16)
24	Leaky ReLU Activation	(1 x 16)
25	Dropout	(1 x 16)
26	Fully Connected (FCs)	(1 x 1)
27	Sigmoid Activation	(1 x 1)

Parameter Math

Standard Convolution Layer:
- Parameters = (kernel size^2 x input channels) x output channels

Separable Depthwise Convolution Layer:
- Depthwise Convolution:
  - Parameters = (kernel size^2) x input channels
- Pointwise Convolution:
  - Parameters = input channels x output channels
- Total:
  - Parameters ≈ Input channels x (kernel size^2 + output channels)

Reduction factor ≈ (1/output channels) + (1/(kernel size^2))
Let N = 512, K = 3, 3
(1/512) + (1/(3^2)) = 0.113 About 9 times less parameters than standard Convolution

Data & Augmentation

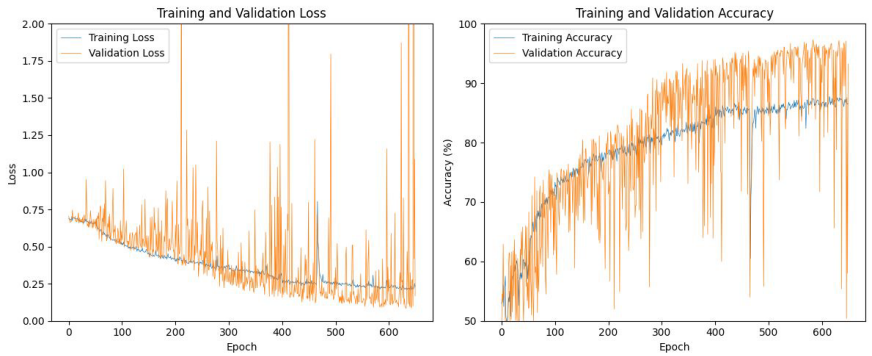
- **Datasets used:**
  - **Real Images:** CelebA & FFHQ datasets
  - **Deepfake Images:** Generated using various GAN architectures - StyleGAN, StarGAN, AttGAN and GDWCT.
- **Data Splits:**
  - **Training:** 60% of the dataset
  - **Validation:** 10% of the dataset
  - **Testing:** 30% of the dataset
  - 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

Deepfake

Real

Training and Validation



Results

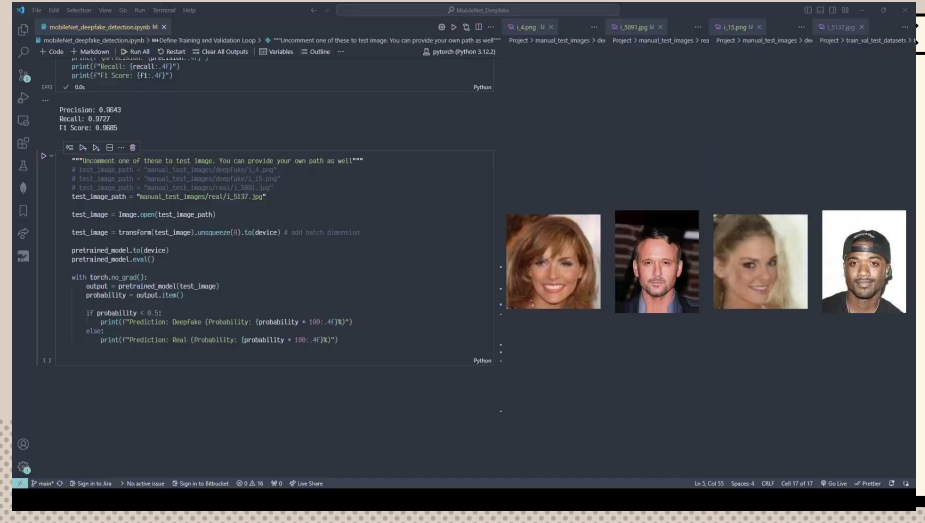
Accuracy
How often model correctly classifies samples

Recall
Proportion of actual positive samples correctly predicted

95.3% | **94.2%** | **96.5%** | **95.4%**

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.