SCSE23-0504

Exploring Lightweight Deep Learning Techniques for Efficient Deepfake Detection

By
Chua Gim Aik
(U2022142B)

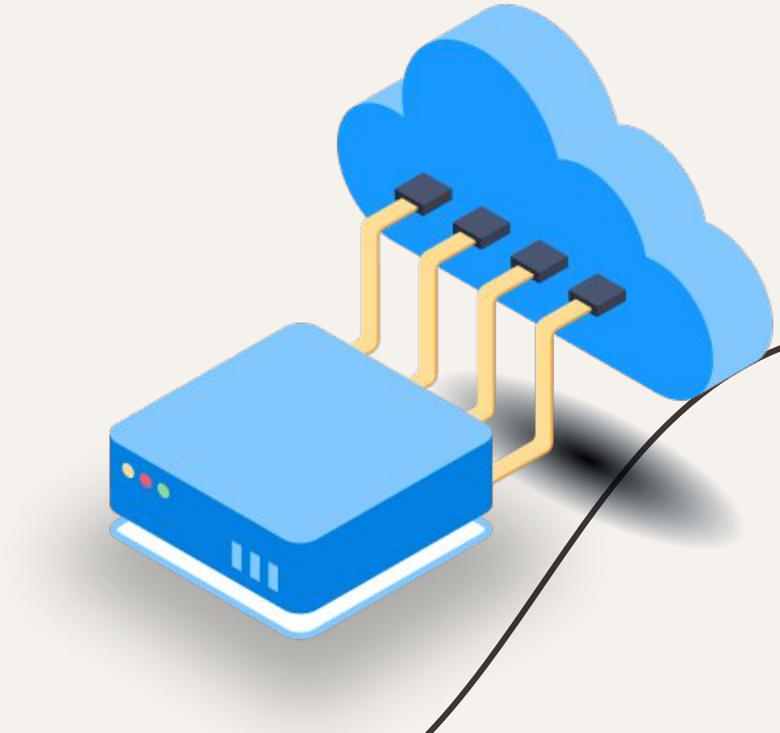
Introducing the emerging Deep Fakes problem

- **Untruthful** media content that imitates a person's **likeness** and depicts them saying or doing things.

- Especially dangerous due to **ease of access** & **exposure** to deep fakes technology, and its potential to threaten livelihood.

Lightweight Deep Learning for deep fake identification

- To combat deep fakes, it is important to be able to **identify** them.
- Cutting-edge detection models come at **high cost** in terms of model size and computing power
- There is a need to push such model towards edge devices like mobile phones due to the **increased exposure risk** and **personal data security concerns**



Project Objective

“Hence, the goal of this project is to explore lightweight deep learning strategies to push effective deep fake detection models towards deployment in weaker computing environments.”

In this project, we…

Developed

An effective strategy to tackle the deep fake video detection problem

Achieved

High detection accuracies through the use of powerful models and training strategies

Optimized

model size and inference time while preserving accuracy by employing efficient model reduction strategies

Chapters

01
Problem Formulation

02
Set-up & Preprocessing

03
Training Strategies

04
Reduction Techniques

05
Demo

06
Conclusion

01

Problem Formulation

Introducing the FF++ Dataset

- Source videos (from Youtube)
- 1000 videos

- Augmented videos (5 different techniques)
- 5000 videos

Defining the problem

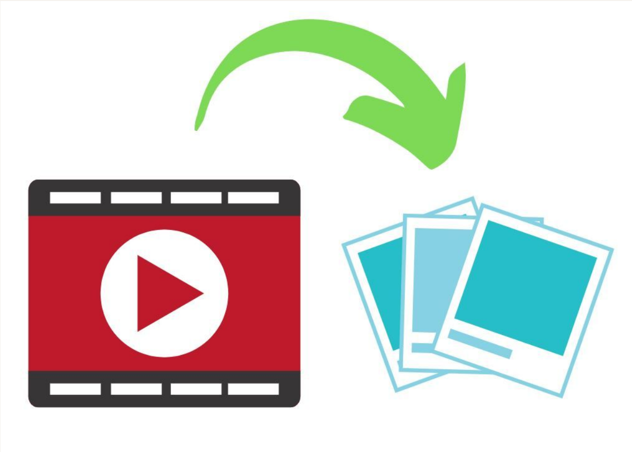
Problem reduction

- Video detection → Image classification

Binary Classification

- 0 (real) vs 1 (fake)
- Model loss function: Binary Cross-entropy Loss (BCE Loss)

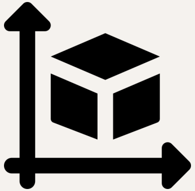
$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$



Defining what "Success" means…

Accuracy

The model must attain high accuracy (> 90%).



Size

We must reduce model size considerably from a baseline without sacrificing *too much* accuracy.

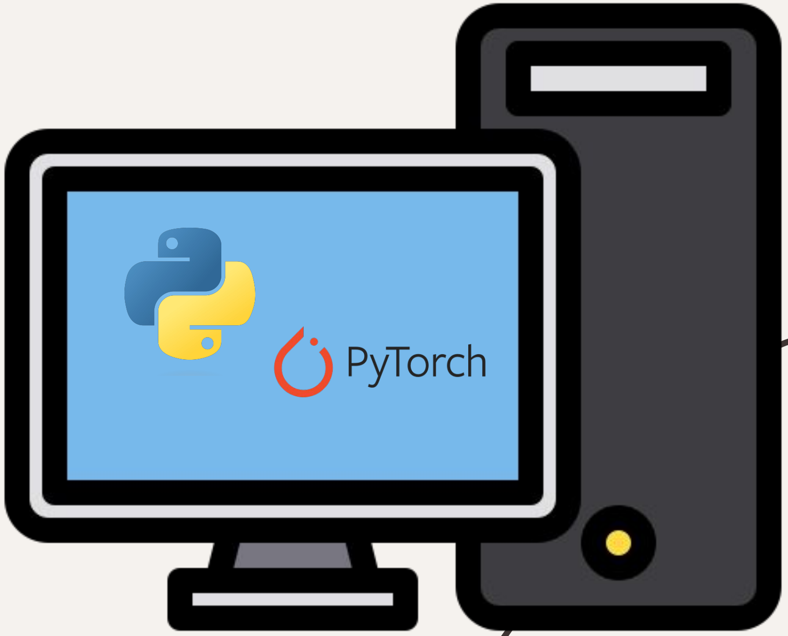
Speed

Our model should enjoy inference speed increments from our proposed changes.

02
Set-up & Preprocessing

Computation Environment

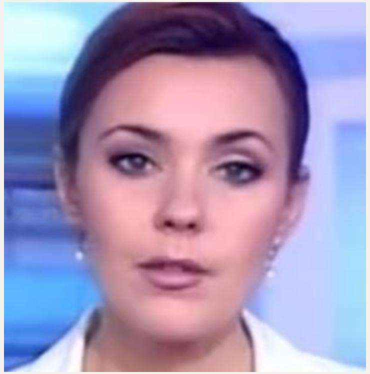
- **CPU** – Intel i7-9700 with a RAM of 32GB
- **GPU** – NVIDIA GeForce RTX 2070 with 8GB memory
- **PyTorch** as the primary deep learning framework
- **Python** – Version 3.10.8



Face detection & Cropping logic (Overview)

Video is broken down into individual frames

OpenCV’s Haar Cascade Face detector detects facial region



Face crop is extracted and transformed into tensor for model input

Face detection & Cropping logic

Handling class imbalance & storage limitation

- For **real** videos (“Youtube”), we extract a frame every 30 frames
    - ~ 17.5K frames
- For **fake** videos (“deepfakes”), we randomly sample 10 frames from every video
    - 10K frames for each deepfake category (50K in total)

Face Detection

- OpenCV’s Haar Cascade Face detection algorithm was used as it was faster than popular alternatives such as MTCNN
- Detected facial region is uniformly expanded to capture surrounding features (e.g. neck, hair, shoulders, etc.)

Face detection & Cropping logic

Data quality control: handling false positives

- **Increased Haar's "Scale Factor"**
  - reduces image size → increasing detection speed
  - detection rate of smaller faces reduced → fewer false positives
- **Increased Haar's "Minimum No. Neighbours"**
  - requires larger number of overlapping detection rectangles
  - hence, increasing detection confidence → fewer false positives
- **Only the biggest face detected is selected for each frame**
  - why: dataset contains mostly videos with a lead subject
  - reduced false positives

False positive example

Consequence: reduced data samples (10% data loss) but remaining is more than sufficient

Data Loading & Transformations

Tensor loading

- Image frames are transformed into pytorch tensors and loaded into CPU memory

Data Sampling

- Real: 10K
- Fake: 15K (3000 for each deepfake method)
- Data is shuffled (to ensure model doesn’t learn order)
- Training-test-validation split: 70-15-15

Augmentation & Regularization

- Random horizontal image flips, image rotations and colour jitters to add noise to data

Category tracing

- Every frame is tagged with their original video category for useful statistics later on

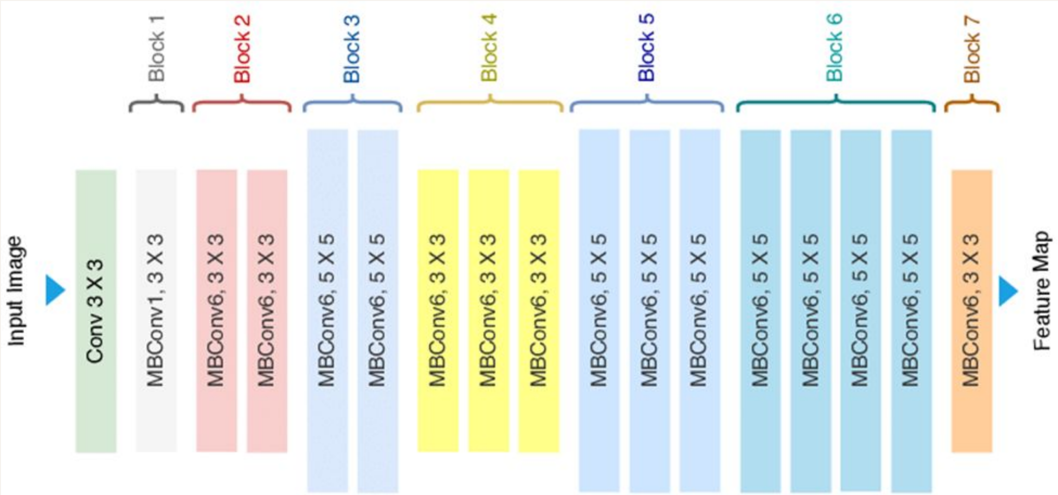
Data Loading & Transformations

Train-test-validation split

Dataset	Sampled	Train	Test	Validation
“Youtube” (Real)	10000	7000	1500	1500
Deep Fakes (5 versions)	15000	10500	2250	2250

03
Training Strategies

EfficientNet - Architecture



- A family of networks designed for efficiency and model performance
- Baseline model “B0” is obtained through Neural Architecture Search
- Primarily made up of Inverted Residual blocks (MBConv)

EfficientNet - Model Scaling

- uniformly scales all dimensions of depth/width/resolution using a compound coefficient (Phi).

$$\text{depth: } d = \alpha^{\phi}$$
$$\text{width: } w = \beta^{\phi}$$
$$\text{resolution: } r = \gamma^{\phi}$$

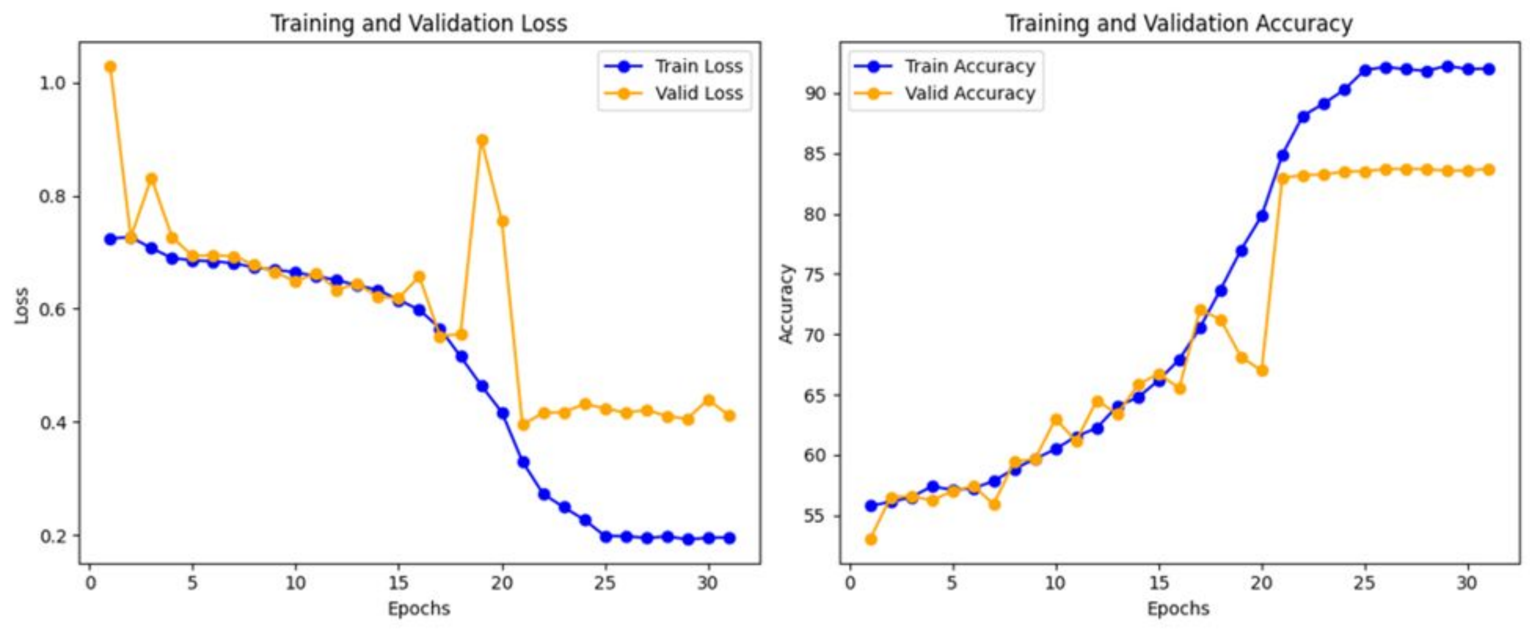
Version	Phi value	Resolution	Drop rate
B0	0	224	0.2
B1	0.5	240	0.2
B2	1	260	0.3
B3	2	300	0.3
B4	3	380	0.4
B5	4	456	0.4
B6	5	528	0.5
B7	6	600	0.5

Training Settings (Overview)

Type	Training	Fine-tuning
No. epochs	30	20
Early stopping patience	5	3
Learning rate	0.0005	0.0002
Weight decay	0.00001	0.00001
Optimizer	SGD	ADAM
Learning Rate Scheduler	Cosine Annealing	None
Warm-up period	10	0
Training time	2 hours+	~ 1 hour

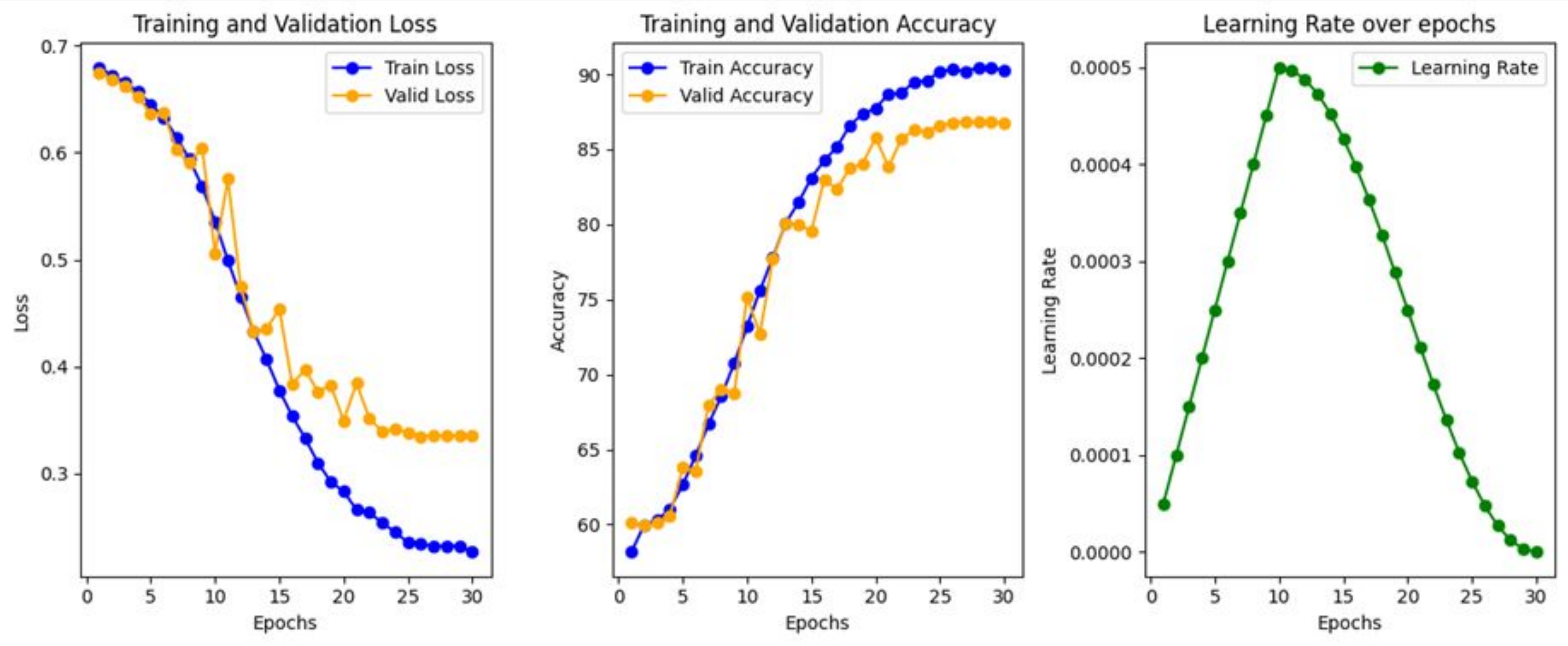
Learning Strategy (No warm-ups or scheduling)

- Unstable + poorer performance



Learning Strategy (Proposed version)

- Improved training stability, convergence and performance



Transfer Learning

WEIGHT INITIALIZATION

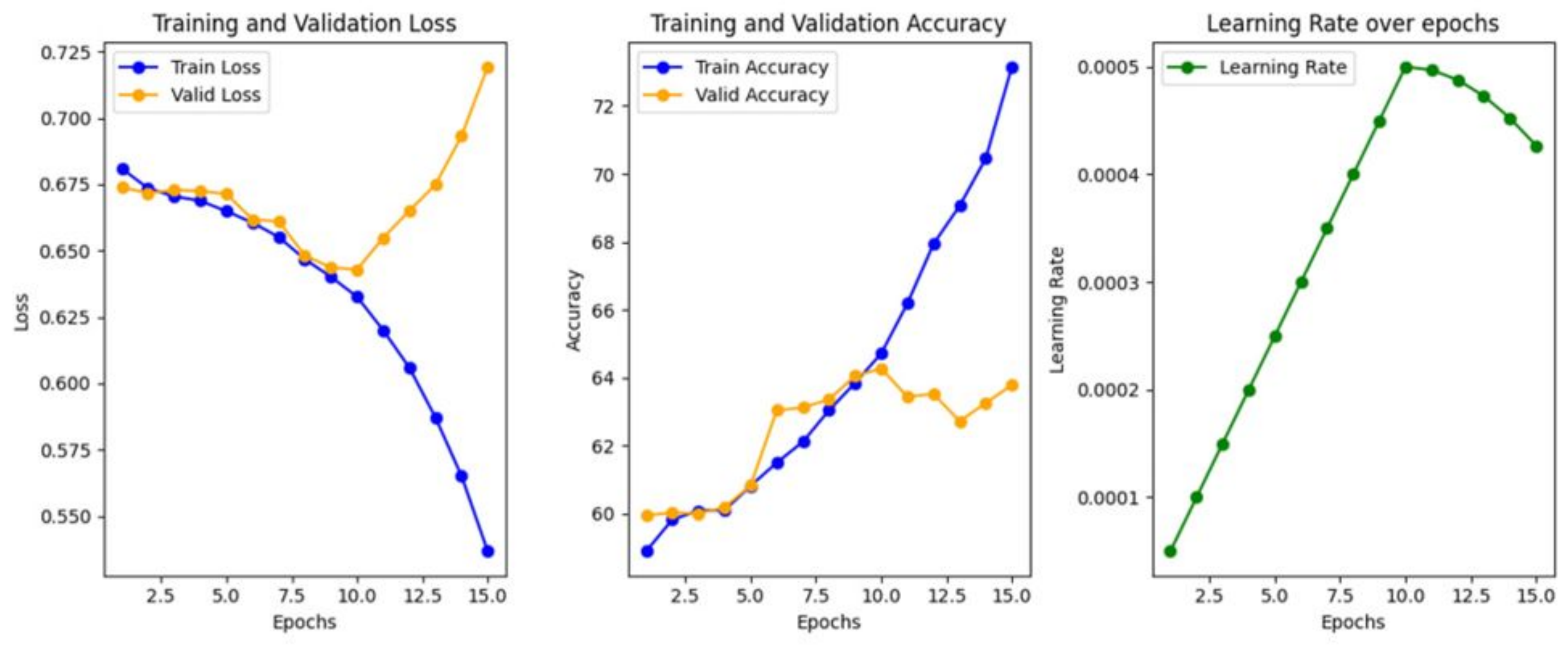
- Model starting point is very important
- Affects quality of training and convergence
- A common practice is to randomize the weight initialization or use techniques like Xavier/Glorot Initialization
- A popular & effective method is to borrow learned weights from a similar task

ImageNet-1k

- Image Classification problem with 1000 classes (e.g. cats, dogs, etc.) and 1.2 million labelled images
- We borrow the learned weights of Efficient Net on the ImageNet-1k dataset and set it as our starting point
- All weights are unfrozen, and the 1-k classifier is replaced with a binary classifier

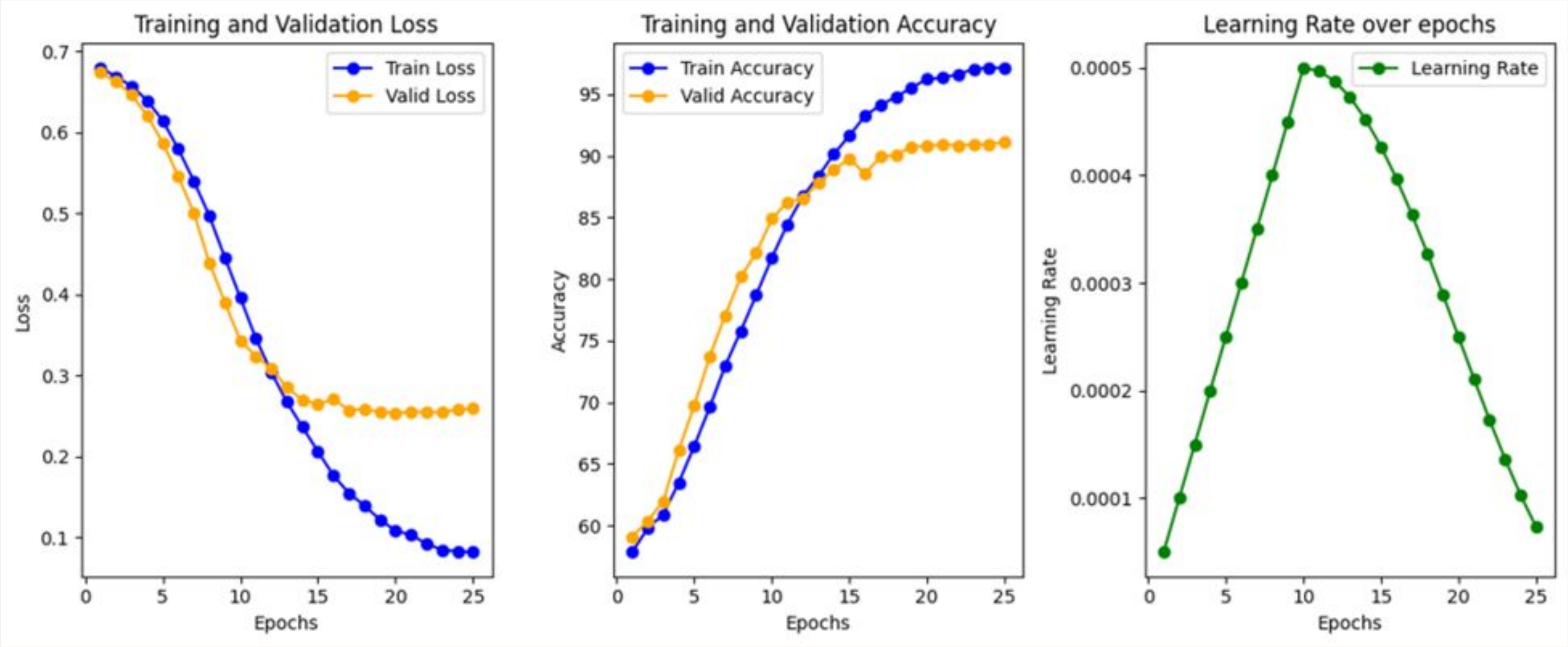
Transfer Learning

- Xavier Initialization - unstable learning observed



Transfer Learning

- ImageNet weight initialization - Better model convergence



Model Scaling Experiment

- **Hardware limitation:** EfficientNet version B3 is the biggest our computing environment can handle
- **Bigger model overfits:** Despite this, B0 was found to be the best overall model (this is likely due to our limited data sample size)

Performance (Acc%) / Model	**EfficientNet-B0**	EfficientNet-B1	EfficientNet-B2	EfficientNet-B3
Overall Acc	**90.03%**	86%	89.52%	86%

Achieving High Accuracy with B0

After some more fine-tuning...

```
Accuracy of the network on tested frames: 94.12 %
Accuracy for class: Deepfakes is 96.90%
Accuracy for class: Face2Face is 95.00%
Accuracy for class: FaceShifter is 92.50%
Accuracy for class: FaceSwap is 93.80%
Accuracy for class: NeuralTextures is 88.30%
Accuracy for class: youtube is 94.94%
```

04
Reduction Techniques

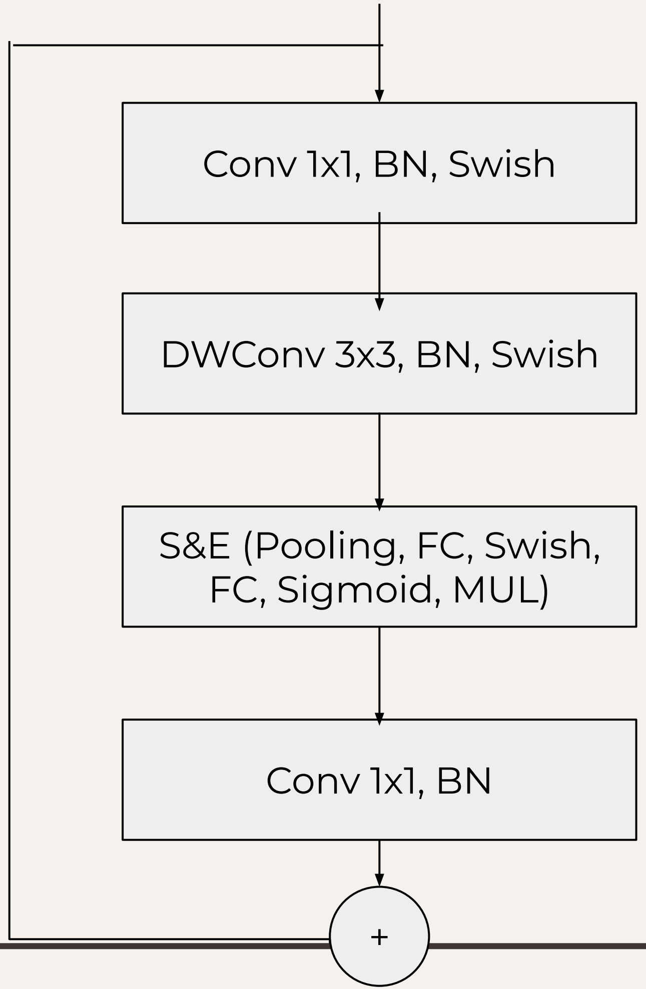
Revisiting EfficientNet

Swish Activation

- Swish function is similar to ReLU except that it allows small negative values when the input is negative, which can help maintain gradient flow
- f(x) = x * sigmoid(x)
- The use of sigmoid increases computation complexity,

Squeeze & Excitation

- Each S&E layer at every MBConv block introduces fully-connected networks (to learn channel statistics)
- This repeated use of S&E increases computation complexity

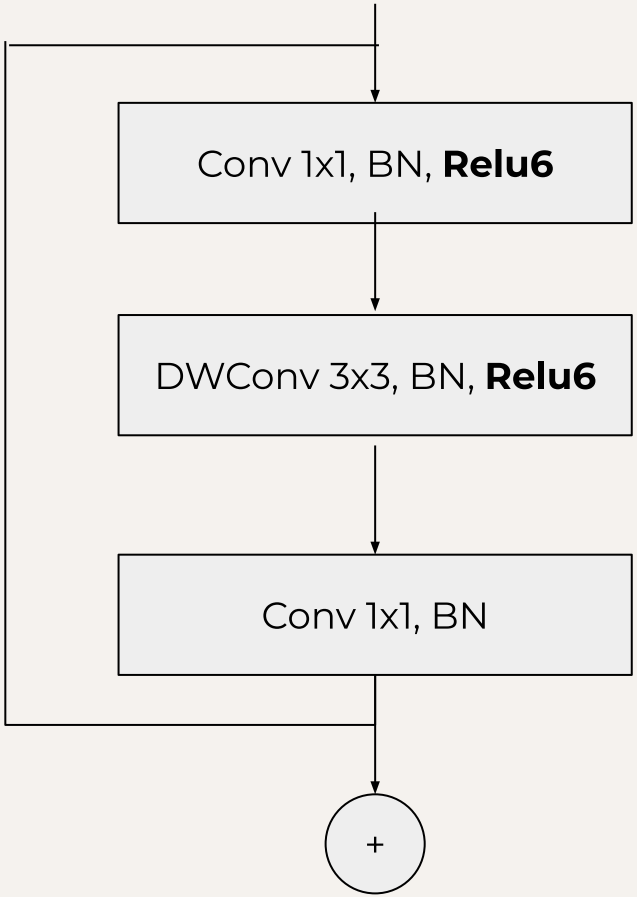


EfficientNet Lite

In 2020, Google proposed removing the aforementioned 2 components to make EfficientNet more edge-device friendly.

Main Changes:

- Swish activation is replaced with the traditional RELU6
- S&E layers are removed



EfficientNet Lite Performance

Metric	EfficientNet-B0	EfficientNet-Lite0
Number of parameters	4,008,829	3,372,289
Size of Model	16.31 MB	13.74 MB
(Avg.) CPU Inference Speed	0.7654 seconds	0.7236 seconds

EfficientNet Lite Performance

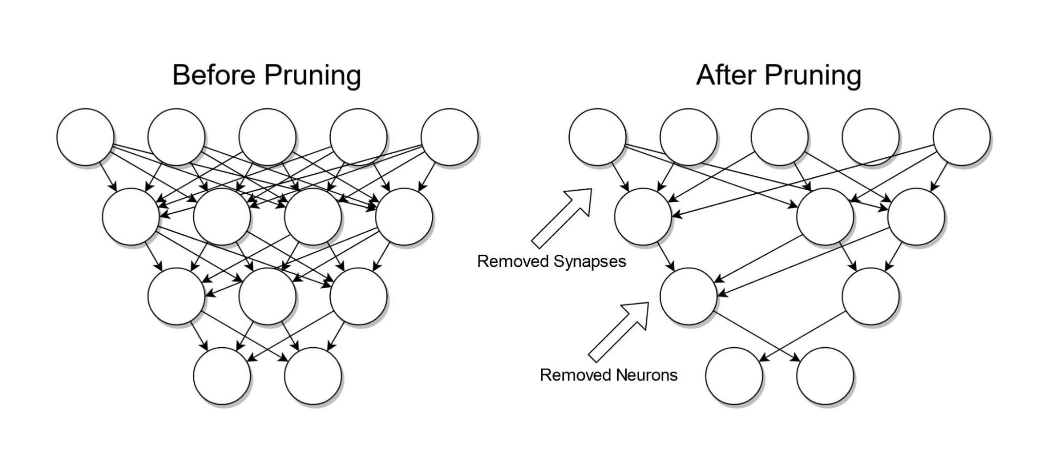
```
Accuracy of the network on tested frames: 90.78 %
Accuracy for class: Deepfakes is 97.20%
Accuracy for class: Face2Face is 96.60%
Accuracy for class: FaceShifter is 92.50%
Accuracy for class: FaceSwap is 92.80%
Accuracy for class: NeuralTextures is 89.10%
Accuracy for class: youtube is 87.92%
```

Model Pruning

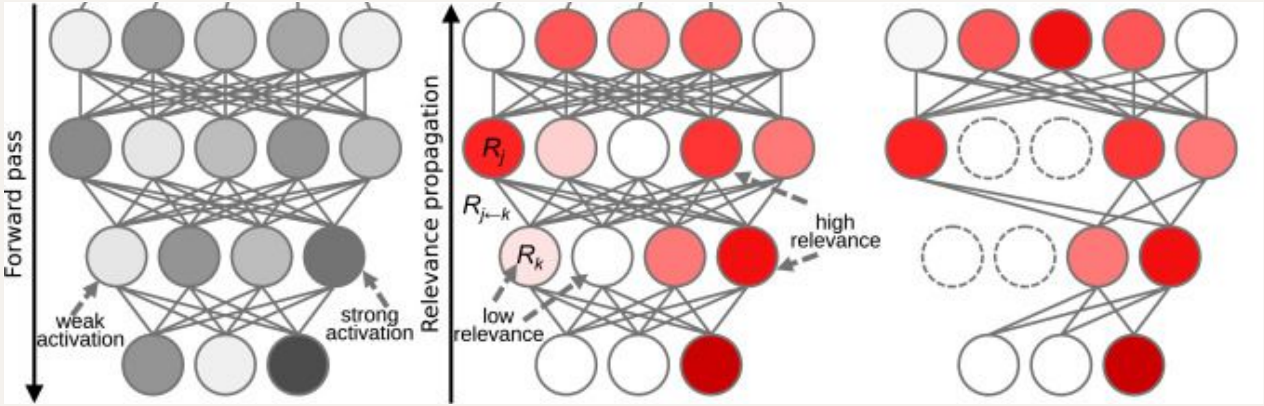
- Not all weights and neurons in the model is useful!

- Pruning is a reduction technique to reduce the less important weights and neurons

- After pruning, fine-tuning is performed to recover loss accuracy



Taylor's Importance Pruning



- This metric helps the pruner approximate the change in the model's output concerning each neuron's activation.
- To employ this, we feed sample input images into the network during pruning and have the pruner evaluate the importance of each neuron and connection

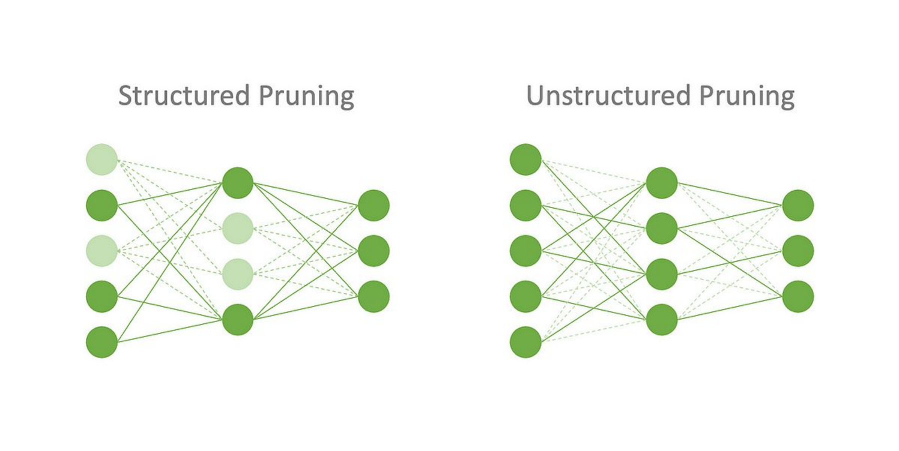
Structured VS Unstructured Pruning

Structured Pruning

- Network structures such as layers and channels are systematically removed
- Results in actual reduction in model size
- *Our choice of pruning*

Unstructured Pruning

- Less salient connections are set to have 0 weight
- These connections are not actually removed (hence model size remains the same)
- Speed-up requires sparse tensor computation



Pruning dependencies

Sequential pruning

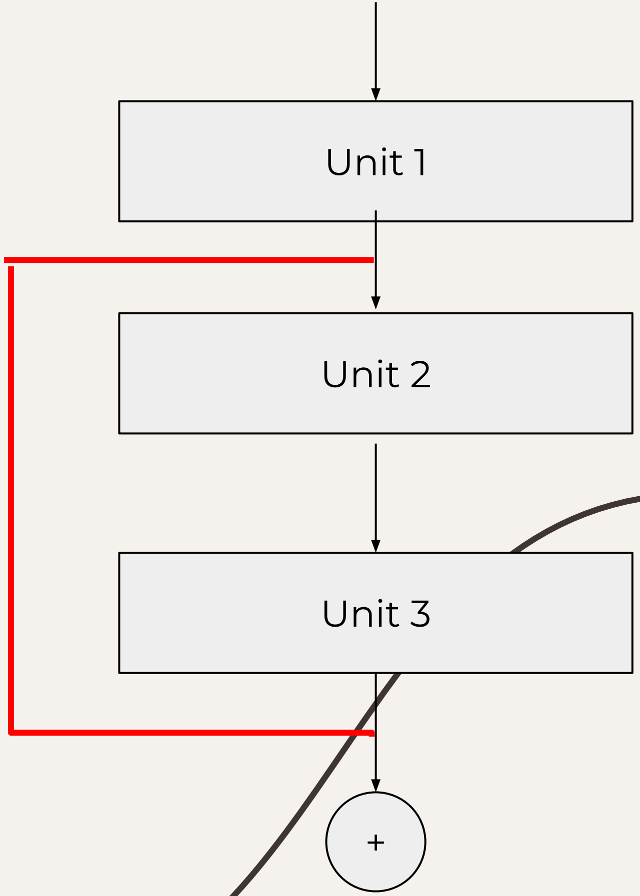
- Trivial, simply propagate unit changes across network to retain network integrity

Non-sequential pruning

- Networks (like EfficientNet) that are non-sequential (e.g. use of residual connections) are tricky
- Unit changes requires future components to change

Solution:

- Grouping components to form "minimal removable units"
- Groups are pruned together to retain network integrity



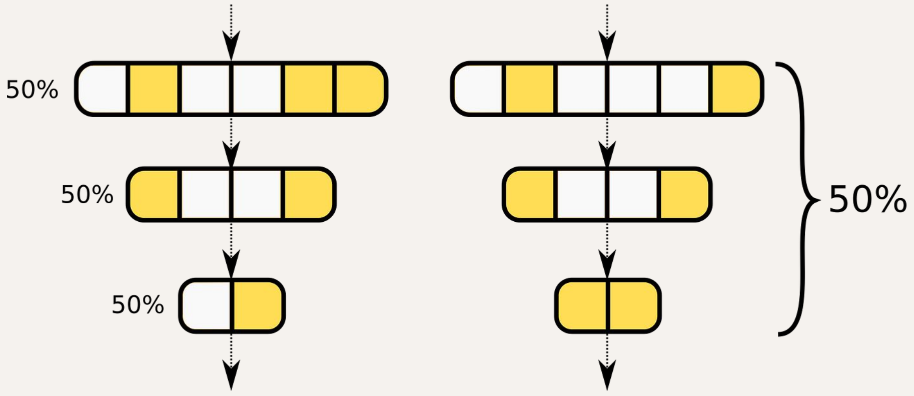
Global VS Local Pruning

Local Pruning

- Every layer is locally pruned
- Difficult to ascertain which layers should be pruned

Global Pruning

- The entire network is viewed as a whole and pruned
- Caveat: layer collapse
- Theoretically better performance



Global VS Local Pruning

5% Local Pruning + FT

```
Accuracy of the network on tested frames: 73.52 %
Accuracy for class: Deepfakes is 76.44%
Accuracy for class: Face2Face is 67.11%
Accuracy for class: FaceShifter is 47.33%
Accuracy for class: FaceSwap is 46.00%
Accuracy for class: NeuralTextures is 52.89%
Accuracy for class: youtube is 96.87%
```

5% Global Pruning + FT

```
Accuracy of the network on tested frames: 90.35 %
Accuracy for class: Deepfakes is 97.78%
Accuracy for class: Face2Face is 96.67%
Accuracy for class: FaceShifter is 91.56%
Accuracy for class: FaceSwap is 92.44%
Accuracy for class: NeuralTextures is 91.11%
Accuracy for class: youtube is 85.00%
```

Global pruning is more effective due to the use of Taylor's Importance - i.e. the weakest contributors across the whole network is pruned without care of their originating layer.

One-shot VS Iterative Pruning

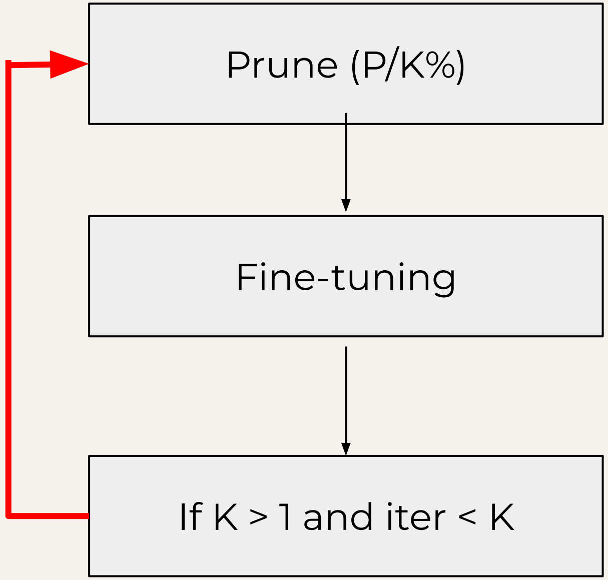
Let pruning ratio = P; K = no. pruning iters

One-shot pruning

- Prune P% and fine-tune once
- i.e. K = 1

K-Iterative Pruning

- Repeat the pruning (P' = P/K%) and fine-tuning K times until pruned percentage is satisfied
- E.g. (if P = 20%, then model is pruned ~4% at each pruning-fine-tuning step for 5 iterations)
- Takes ~K-times longer



1-shot VS 5-shot Pruning

1-shot Pruning (20%)

```
Accuracy of the network on tested frames: 88.29 %
Accuracy for class: Deepfakes is 97.11%
Accuracy for class: Face2Face is 92.89%
Accuracy for class: FaceShifter is 90.00%
Accuracy for class: FaceSwap is 85.11%
Accuracy for class: NeuralTextures is 88.44%
Accuracy for class: youtube is 84.67%
```

5-shot Pruning (20%)

```
Accuracy of the network on tested frames: 90.00 %
Accuracy for class: Deepfakes is 97.56%
Accuracy for class: Face2Face is 95.78%
Accuracy for class: FaceShifter is 92.22%
Accuracy for class: FaceSwap is 90.22%
Accuracy for class: NeuralTextures is 89.33%
Accuracy for class: youtube is 85.47%
```

Iterative pruning takes longer but yields accuracy increments as pruner gets to re-evaluate importance at each pruning step

Fruits of our pruning strategy

30% pruning is the maximum we can go! Model size and inference time **cut by half!**

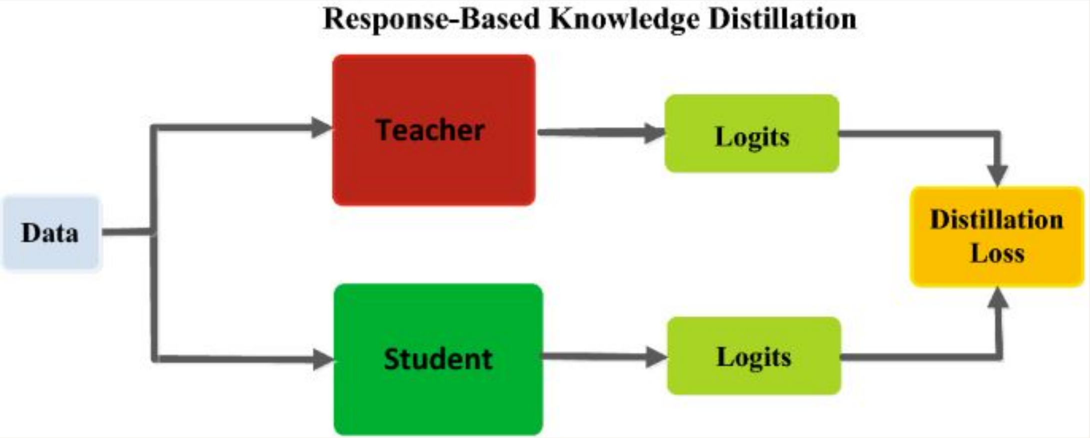
Prune ratio	Model Accuracy	Model Size	Model Parameters	Inference Speed (CPU)
20%	90%	10.33 MB	2,544,127	0.4509s
25%	89.68%	9.57 MB	2,339,132	0.4248s
30%	**88.77%**	**8.81 MB**	**2,151,361**	**0.3935s**
35%	78.61%	8.07 MB	1,968,970	0.3305s

Knowledge Distillation

Recovering lost accuracy

- KD employs the help of a larger, more superior teacher model to pass on knowledge to our model of interest (student)

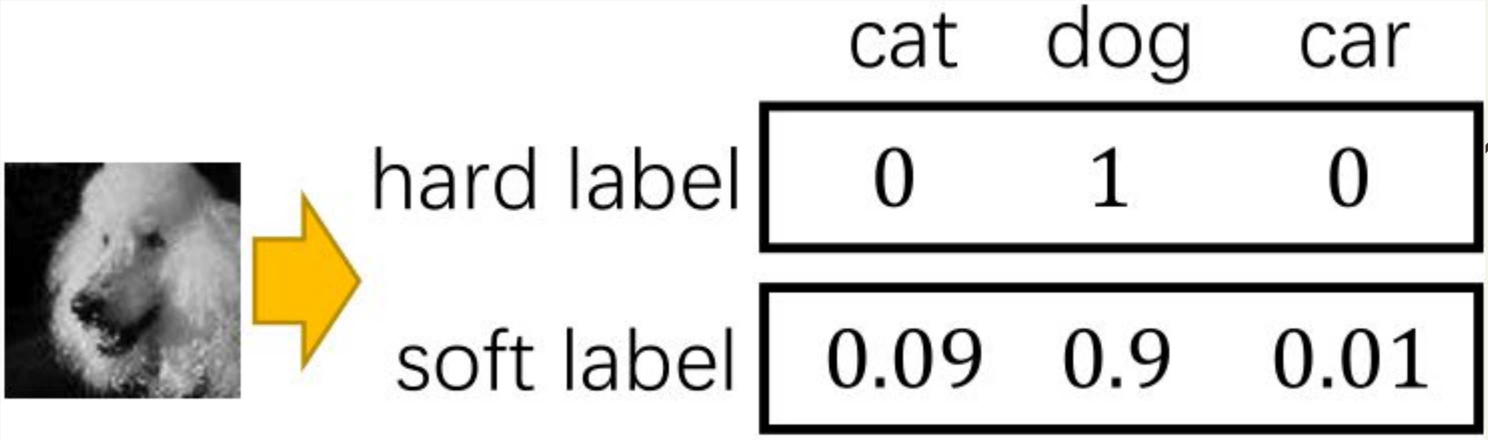
- By teaching the teacher's loss to the student, the student model can attempt to replicate the teacher's internal representation



Knowledge Distillation

Soft VS hard label loss

- Soft labels *contain more information* (why did the teacher think the image was 10% fake and 90% real?)



KD Formulation & Training Augmentation

Our KD Formulation

$$L_{KD} = \alpha * soft\ label\ loss + (1 - \alpha) * hard\ label\ loss$$

Teacher Model Selection

- EfficientNet B0 is our baseline model and it has the highest accuracy
- It is fitting to teach our LITE and PRUNED models

Training logic augmentation

- Teacher model's weights is frozen
- At each training step (mini-batch), the teacher model performs predictions and its output logits is extracted for the student's learning

Knowledge-distilled Training (Optimal Alpha)

Alpha value	0	**0.25**	0.5	**0.75**	1
Accuracy (%)	87.68	**91.44**	90.16	**92.24**	90.03

- Knowledge distillation on LITE0 models with NO fine-tuning
- **Optimal alpha seems to be 25% KD or 75% KD**
- Aside from accuracy, these ratios had smoother training and better model convergence

25% KD vs 75% KD (after fine-tuning)

LITE0 w/ 25% KD

```
Accuracy of the network on tested frames: 93.25 %
Accuracy for class: Deepfakes is 98.22%
Accuracy for class: Face2Face is 97.33%
Accuracy for class: FaceShifter is 97.11%
Accuracy for class: FaceSwap is 93.33%
Accuracy for class: NeuralTextures is 89.33%
Accuracy for class: youtube is 90.53%
```

LITE0 w/75% KD

```
Accuracy of the network on tested frames: 92.91 %
Accuracy for class: Deepfakes is 98.00%
Accuracy for class: Face2Face is 97.78%
Accuracy for class: FaceShifter is 93.78%
Accuracy for class: FaceSwap is 95.11%
Accuracy for class: NeuralTextures is 89.33%
Accuracy for class: youtube is 90.07%
```

Both are very close in terms of performance, but 25% KD comes out ahead.

Interestingly, with KD - **training time improved as KD promoted stable training and model convergence**.

Knowledge-distilled Training Results

Successfully recovered some lost accuracy from LITE0 changes!

EfficientNet-B0	EfficientNet-Lite0	**Lite0 + 25% KD**
94.12%	90.78%	**93.25%**

Knowledge-distilled Pruning

Similar to KD training, we employ 25% KD at each fine-tuning step during iterative pruning.

KD-based pruning allowed us to prune 5% than before!

Prune ratio	Model Accuracy	Model Size	Model Parameters	Inference Speed (CPU)
25%	93.71%	9.5 MB	2,322,717	0.4175s
30%	93.07%	8.78 MB	2,144,299	0.3866s
35%	**91.89%**	**7.97 MB**	**1,942,167**	**0.3699s**
40%	78.53%	7.21 MB	1,754,377	0.2645s

We've came a long way…

Using our proposed training and reduction strategies, we have obtained **high accuracy** and **significantly reduced model size and inference time by more than half!**

Model	Prune ratio	Accuracy	Size	Parameters	Inference time
(Baseline) B0	0%	94.12%	16.31 MB	4,008,829	0.7654s
Lite0	0%	90.78%	13.74 MB	3,372,289	0.7236s
Lite0	30%	88.77%	8.81 MB	2,151,361	0.3935s
Lite0 + KD	30%	93.07%	8.78 MB	2,144,299	0.3866s
Lite0 + KD	**35%**	**91.89%**	**7.97 MB**	**1,942,167**	**0.3699s**

05
Demo

Finishing what we started

Currently, we only solved the image classification problem. For completeness, we propose a strategy for **deep fake detection at the video level.**

1) Extract frames & detect face crops

2) Apply model prediction

3) Splice frames together to form video

This allows the model to continue operating as an image classifier

The model returns the binary classification output. Bounding boxes are injected to denote the nature of detected faces (i.e. “real” or “fake”)

Merge the augmented frames back into the original video. Determine the video authenticity by the rate of “deepfake” and “real” classification.

Web Demo - Deepfake detection

06
Conclusion

Summary Findings

- We developed a comprehensive deep fake detection solution at the video level
- We employed training strategies that helped us achieve high deep fake detection accuracy
- We showed how model reduction strategies can help reduce model cost while preserving model performance

Limitations & Future works

Frame processing bottleneck

Frame extraction, face detection and cropping takes up > 50% of the solution run time. Steps should be taken to reduce this upfront cost.

Deployment on edge devices

Limitation: current solution is still far too slow to be deployed on edge devices.

Quantization

Quantization maps floating-point weights into integer, reducing model weights by ~x4 at the cost of accuracy. This is quintessential in edge devices.

Data variety

Although FF++ includes multiple deepfake techniques, the model is unlikely to perform well on unseen & more advanced/novel deepfake methods. More training data and variety is required.

Thank you!

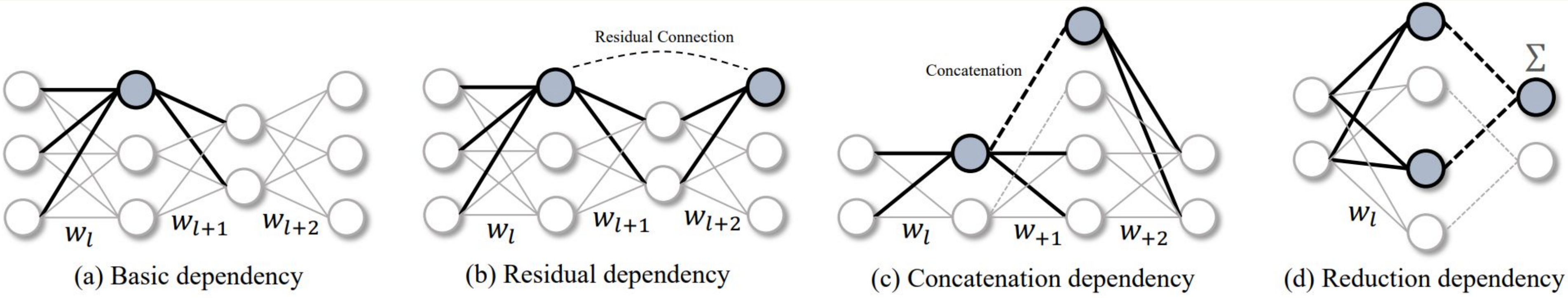


Figure 2. Grouped parameters with inter-dependency in different structures. All highlighted parameters must be pruned simultaneously.