Deepfake Detection

CS517 EndSem-Project

Arnav Kharbanda(2021CSB1072)
Keshav Aggarwal(2021CSB1104)
Yashasav Prajapati(2021CSB1143)

References

- Python Machine Learning Book
- Face Deepfake Detection Challenge

Doctored Video of Nancy Pelosi

- In 2019, a video of Nancy Pelosi, the Speaker of the United States House of Representatives was widely circulated.
- The video was slowed down to make her speech appear slurred and drunken, and it was shared widely on social media platforms.
- While the video was not a sophisticated deepfake, it highlighted the potential for manipulated media to be used to spread disinformation and damage the reputations of public figures. The incident raised concerns about the potential impact of more advanced deepfake technology on politics, public opinion, and the media.

Since then, deepfakes have created several problems of this kind, ranging from political manipulation to the propagation of misinformation including in pornography.

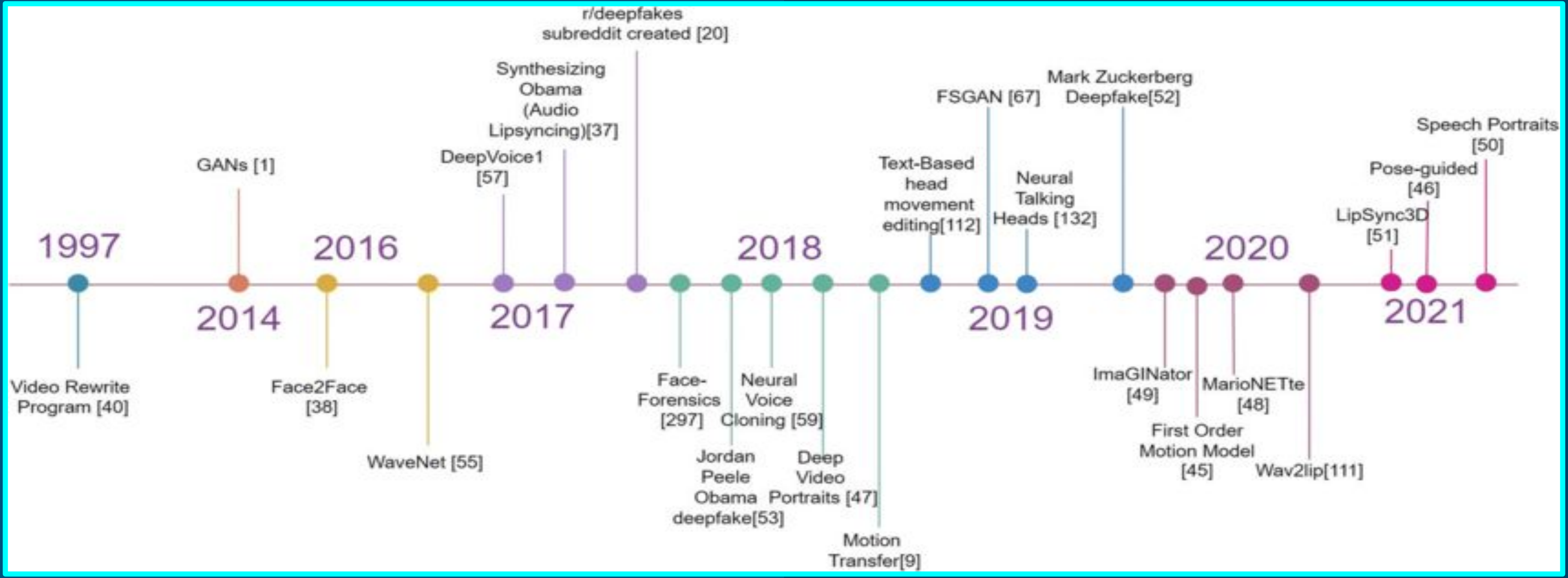
<u>This clearly shows the reason we need Deepfake Detection</u>

Source: CBS News

Problem Statement

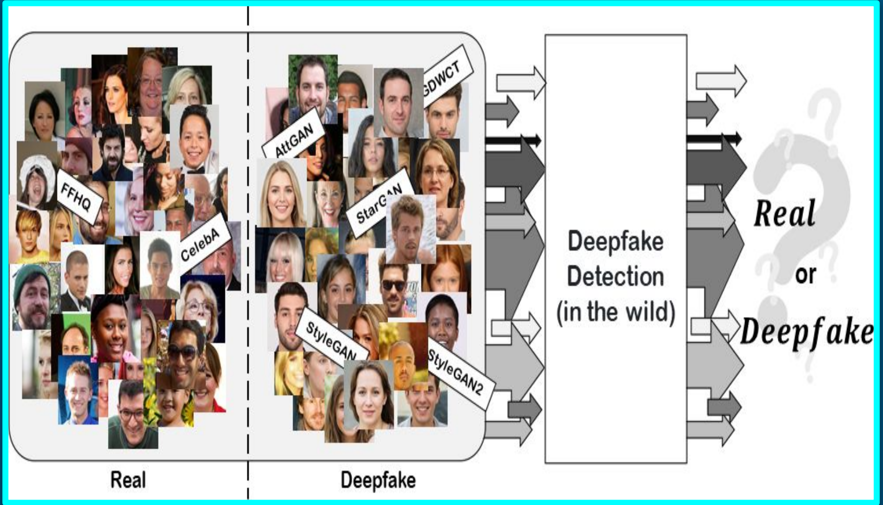
Deepfakes pose a significant threat to society. Existing solutions for detecting them lack generalizability, making them ineffective in real-world situations. **To address this issue, we aimed to develop robust and generalizable Deepfake detection model that can effectively identify Deepfakes in real-world scenarios.** Specifically, our aim is to focus on Deepfake images of human faces, presenting an opportunity to create solutions that can counteract the malicious use of this technology.

Timeline of evolution of Deepfakes

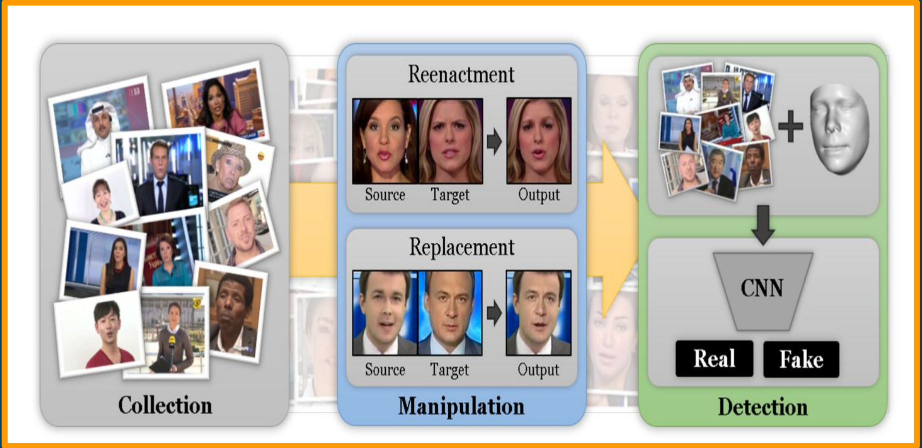


Source: ResearchGate

Sample Model



Source: [Deepfake challenge]



Source: [Deepfake detection]

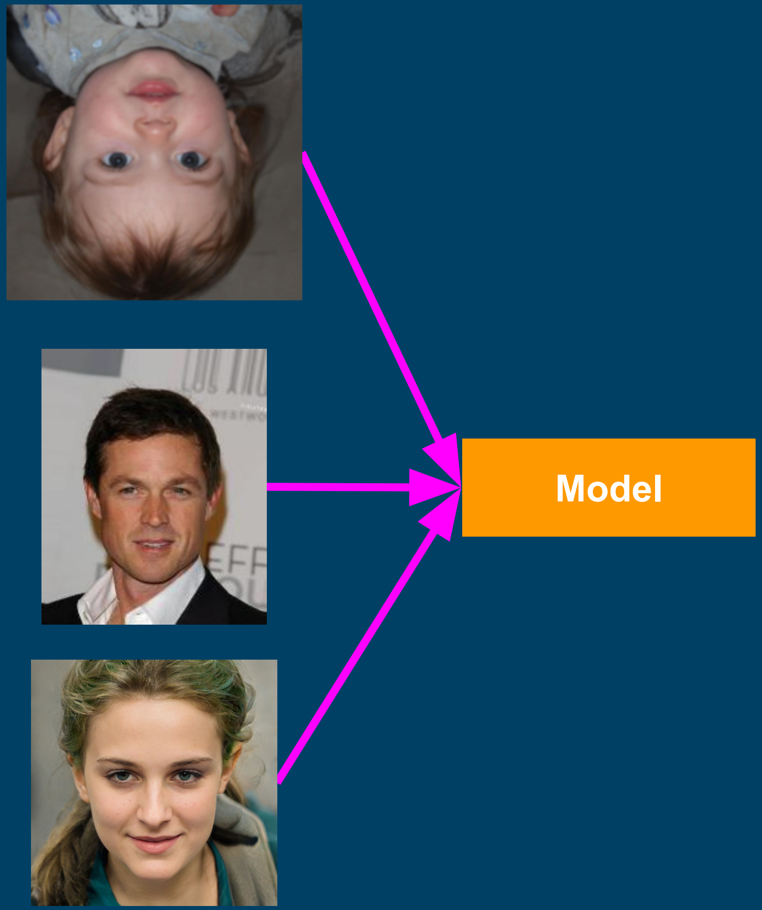
INPUT

Query Image

The Deepfake detection model takes a single input image and pass it through model having different convolution and pooling layers.

ILLUSTRATION

INPUTS



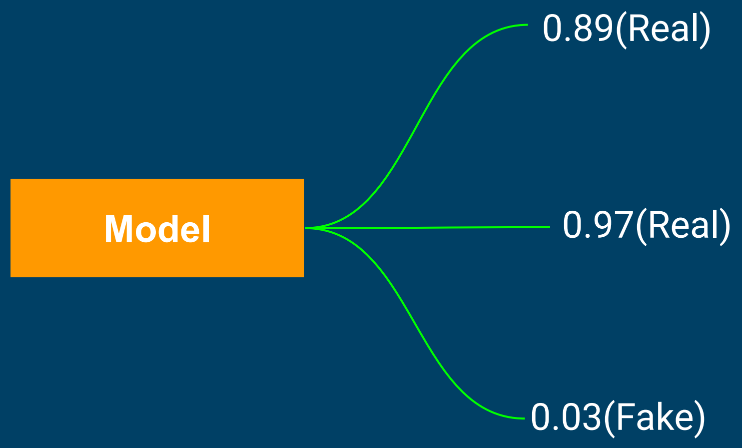
OUTPUT

Prediction Value

- The output of the system is a prediction value, this value tells whether the image is a deepfake or not.
- True positive images are those that are correctly identified as a deepfake by the model, that is, they are manipulated or synthetic created, and the model was able to detect the presence of manipulations or inconsistencies that are indicative of a deepfake.
- False positive images are those that are incorrectly identified as a deepfake by the model, when in fact it is a genuine or real image.

ILLUSTRATION

OUTPUTS



DataSets Used

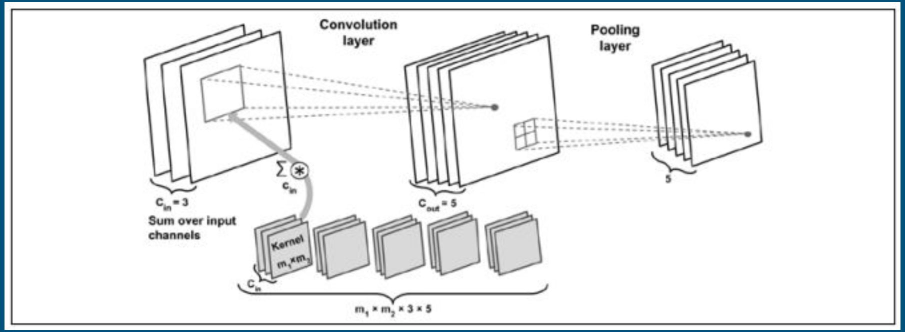
- Celeb-A(Real Images)
- FFHQ(Real Images)
- ATTGAN(GAN Images)
- GDWCT(GAN Images)
- StarGAN(GAN Images)
- STYLEGAN(GAN Images)
- STYLEGAN2(GAN Images)

Evaluation Metrics

- Accuracy = (TP + TN) / (TP + TN + FP + FN)
- Precision = TP / (TP + FP)
- Recall = TP / (TP + FN)
- F1-Score = 2 * Precision * Recall / (Precision + Recall)
- Specificity = TN / (TN + FP), and
- AUC-ROC can be calculated by plotting the Receiver Operating Characteristic (ROC) curve and computing the area under the curve.

Where,
TP = True Positives,
TN = True Negatives,
FP = False Positives,
FN = False Negatives

What was the solution approach?



The general solution has following steps:

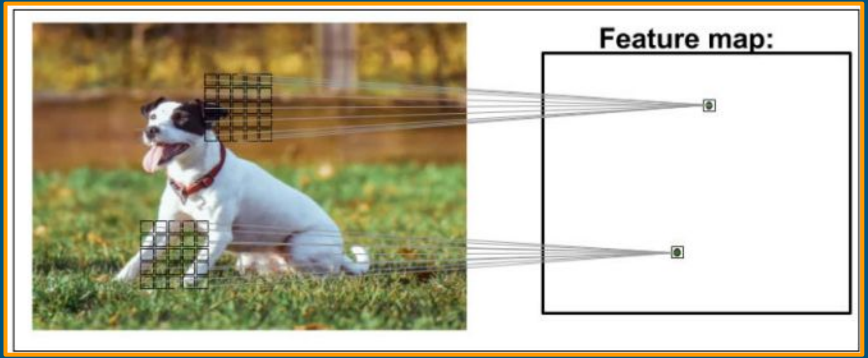
1. **Data Augmentation** - Generating new data for model training.
2. **CNN** - Convolutional Neural Network for image analysis
3. **Dropout** - Preventing Overfitting in neural networks
4. **Pooling** - Downsampling to extract key information
5. **Evaluation** - Assessing model performance on test data

Hyperparameters Used

Batch Size: 128
Epochs: Values ranging from 100 to 1000, applied Early Stopping to prevent overfitting and unnecessary training
Learning Rate: 0.01 for Adam Optimizer

CNN better on image-related works

CNN usually computes feature maps form an input image, where each element comes from a local patch of pixels in the input image:



This local patch of pixels is referred to as the **local receptive field**.

Two important ideas supporting the CNNs:

- **Sparse connectivity**: A single element in the feature map is connected to only a small patch of pixels. (This is very different from connecting to the whole input image as in the case of perceptrons.)

- **Parameter-sharing**: The same weights are used for different patches of the input image.

Data Augmentation

Data augmentation is a set of techniques to artificially increase the amount of data by generating new data points from existing data.

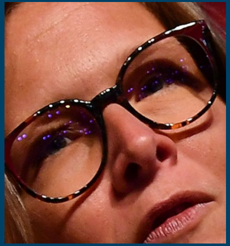
Original Image

Flipped Image

GrayScale Image

Channel Shuffle

Image with Impurities



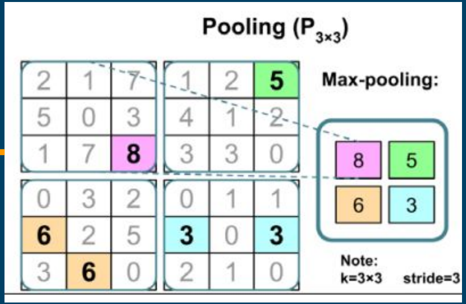
Rotated Image

Some Operations used in the model

Discrete Convolution in 2D

$$\boldsymbol{Y} = \boldsymbol{X} * \boldsymbol{W} \rightarrow \quad Y[i,j] = \sum_{k_1=-\infty}^{+\infty} \sum_{k_2=-\infty}^{+\infty} X[i-k_1, j-k_2]\, W[k_1, k_2]$$

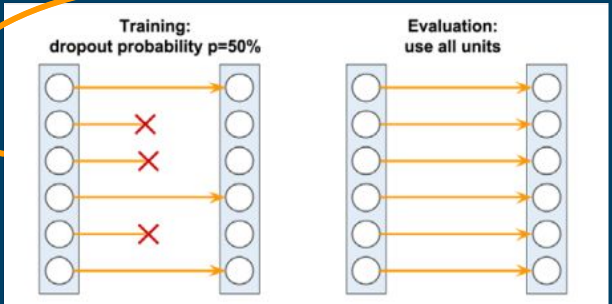
Pooling (max-pooling) introduces a local invariance. This means that small changes in a local neighborhood do not change the result of max-pooling. Therefore, it helps with generating features that are more robust to noise in the input data



Loss function

Loss function	Usage	Examples	
		Using probabilities `from_logits=False`	Using logits `from_logits=True`
BinaryCrossentropy	Binary classification	y_true: 1 y_pred: 0.69	y_true: 1 y_pred: 0.8

This random dropout can effectively prevent overfitting



Building Dataset pipeline for training and validation

Using tensorflow with keras for training

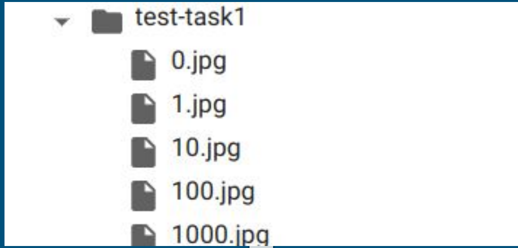
- train
  - fake
  - real

Training data containing data with 2 labels

```python

ds = tf.keras.utils.image_dataset_from_directory(
    '/content/sample_data/train',
    labels='inferred',
    label_mode='binary',
    class_names=None,
    color_mode='rgb',
    batch_size=128,
    image_size=(256, 256),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation='bilinear',
    follow_links=False,
    crop_to_aspect_ratio=False
)

train_ds = ds.take(40)
val_ds = ds.skip(7)

```

Building Dataset pipeline for training and validation

Using tensorflow with keras for testing



Testing data contains several images with their correct corresponding labels in another text file

```python
test_ds = tf.keras.utils.image_dataset_from_directory(
    '/content/sample_data/test-task1',
    labels=Labels,
    label_mode='binary',
    class_names=None,
    color_mode='rgb',
    batch_size=128,
    image_size=(256, 256),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation='bilinear',
    follow_links=False,
    crop_to_aspect_ratio=False
)
```

Model Building and Compiling

```
model = tf.keras.Sequential([
  resize_and_rescale,
  data_augmentation,
  tf.keras.layers.Conv2D(
      32, (3, 3), padding='same', activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),
  tf.keras.layers.Dropout(rate=0.5),
  tf.keras.layers.Conv2D(
      64, (3, 3), padding='same', activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),
  tf.keras.layers.Dropout(rate=0.5),

  tf.keras.layers.Conv2D(
      128, (3, 3), padding='same', activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),
  tf.keras.layers.Dropout(rate=0.5),
  tf.keras.layers.Conv2D(
      256, (3, 3), padding='same', activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Conv2D(
      512, (3, 3), padding='same', activation='relu'),
  tf.keras.layers.GlobalAveragePooling2D(),
  tf.keras.layers.Dense(1, activation=None)
])
```

Model made using tensorflow with keras, with several layers or Convolution and Pooling

Building and compiling the model, with some loss functions and optimizers

```
model.compute_output_shape(input_shape=(None, 64, 64, 3))
model.build(input_shape=(None, 64, 64, 3))
model.compile(optimizer=tf.keras.optimizers.Adam(),
   loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
   metrics=['accuracy']
)
```

Model Training

```
history = model.fit(train_ds, validation_data=val_ds, callbacks=[checkpoint, es],
  epochs=1000)
model.save('/content/model')
```

Training the model and saving it at the end

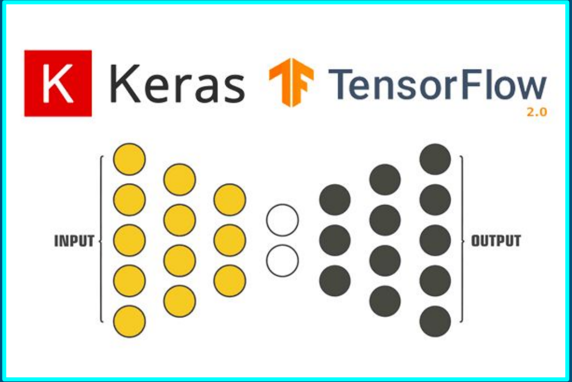
Training

```
Epoch 1/1000
40/40 [==============================] - ETA: 0s - loss: 0.7131 - accuracy: 0.4418
Epoch 1: val_accuracy improved from -inf to 0.33324, saving model to checkpoints
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolutio
40/40 [==============================] - 738s 18s/step - loss: 0.7131 - accuracy: 0.4418 - val_loss: 0.6719 - val_accuracy: 0.3332
Epoch 2/1000
40/40 [==============================] - ETA: 0s - loss: 0.6457 - accuracy: 0.5834
Epoch 2: val_accuracy improved from 0.33324 to 0.33332, saving model to checkpoints
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolutio
40/40 [==============================] - 715s 18s/step - loss: 0.6457 - accuracy: 0.5834 - val_loss: 0.6671 - val_accuracy: 0.3333
Epoch 3/1000
40/40 [==============================] - ETA: 0s - loss: 0.6407 - accuracy: 0.5836
Epoch 3: val_accuracy improved from 0.33332 to 0.33745, saving model to checkpoints
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolutio
40/40 [==============================] - 702s 18s/step - loss: 0.6407 - accuracy: 0.5836 - val_loss: 0.6531 - val_accuracy: 0.3374
Epoch 4/1000
40/40 [==============================] - ETA: 0s - loss: 0.6349 - accuracy: 0.6432
Epoch 4: val_accuracy improved from 0.33745 to 0.40732, saving model to checkpoints
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolutio
40/40 [==============================] - 732s 19s/step - loss: 0.6349 - accuracy: 0.6432 - val_loss: 0.6443 - val_accuracy: 0.4073
```

Tensorflow and Keras

- TensorFlow is a low-level, flexible library for numerical computation that allows developers to create complex machine learning models.

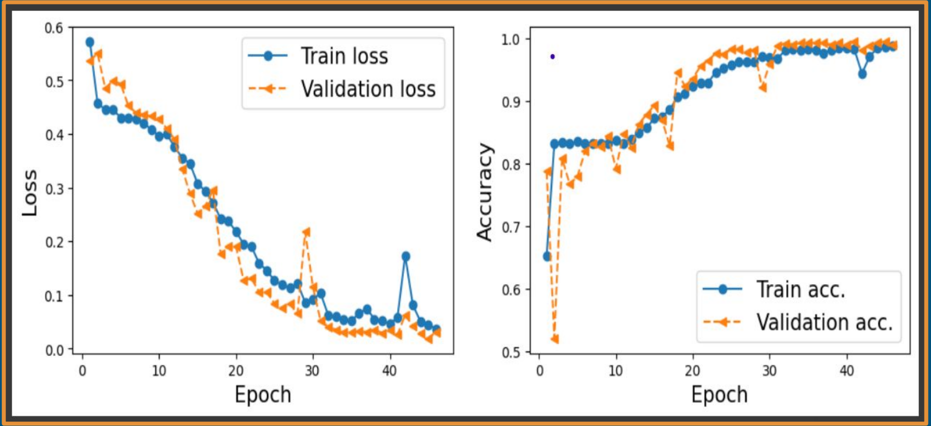
- Keras, on the other hand, is a high-level neural network API that is built on top of TensorFlow. It provides a simple and easy-to-use interface for building and training deep learning models. Keras is widely used by developers and researchers due to its ease of use and ability to quickly prototype models.

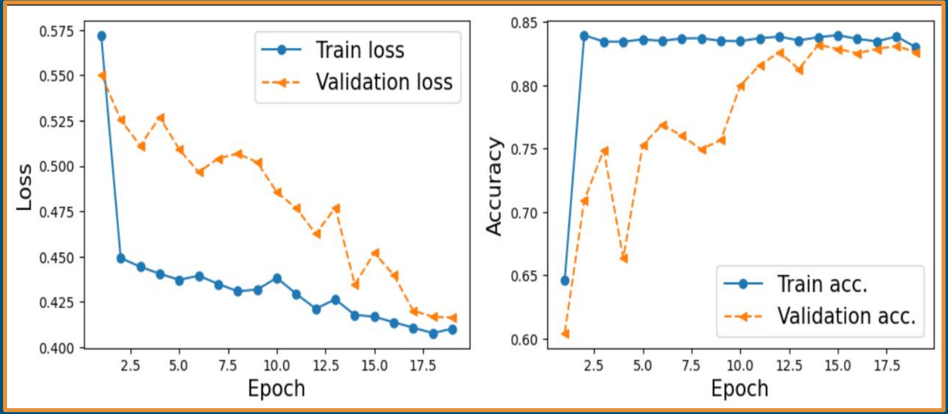


Comparison among different data sets.

Results with small data set

Results with bigger (augmented) data set





Results with augmented data set gets improved by 10% accuracy.

RESULTS

```
model.evaluate(test_ds)
```

```
55/55 [==============================] - 28s 428ms/step - loss: 0.6457 - accuracy: 0.6720
[0.6457478404045105, 0.671999990940094]
```

Evaluation Metrics

```
[24] metric = model.predict(test_ds.take(1), verbose=1)
```

```
1/1 [==============================] - 16s 16s/step
```

```
print(metric)
```

```
[[1.7453243 ]
 [0.89414597]
 [2.6484466 ]
 [2.8712974 ]
 [0.97392946]
 [0.65061766]
 [1.3413068 ]
 [0.38719076]
 [0.6101905 ]
 [0.38366163]
 [1.0419937 ]
 [2.537054  ]
 [0.76276535]
 [0.40314674]
 [1.5444108 ]
 [0.67570114]
 [1.1828473 ]
 [1.7903951 ]
 [1.4690809 ]
 [1.0358201 ]
```

Other Metrics on a subset of images

```
print(precision_score(Labels[0:128], y_pred , average="macro"))
print(recall_score(Labels[0:128], y_pred , average="macro"))
print(f1_score(Labels[0:128], y_pred , average="macro"))
```

```
0.12890625
0.5
0.20496894409937888
```

Output on some images

Learnings

- Multilayers CNNs, Convolution in CNNs, Max-Pooling, Dropout, creating dataset pipelines.
- Data Augmentation
  - Augmented data gives better results and higher accuracy because it learns from several distorted images also and so it identifies the pattern more accurately.
- Hyperparameter tuning: Optimizing hyperparameters such as learning rate, batch size, and number of epochs can lead to better model performance.
  - Tuning the batch size, number of epochs can improve the accuracy of the model.
  - Early Stopping along with Checkpoint saving after some iterations can prevent overfitting or unnecessary training of the model.

Thanks