SIGNATURE VERIFICATION

For what i am learning this

To get the concept of computer vision and image processing with machine learning and what are the various concepts in image processing and gain knowledge in deep learning.Still i know how to process in raw text,after this there are some many things we are going to explore.

Why doing this project

Signature recognition is a biometric method of identifying individuals based on their unique signature patterns. It's widely used for:

- **Authentication**: Validating the legitimacy of a signature.
- **Forgery Detection**: Identifying fraudulent signatures.
- **Secure Transactions**: Ensuring that sensitive operations are performed by authorized personnel.

When this project will be helpful

Imagine a scenario where an organization wants to verify signatures on legal documents. However, manual verification is error-prone and time-consuming. The task is to build an automated system capable of:

1. Identifying the owner of a given signature.
2. Verifying whether a signature is genuine or forged.

What need to be learned to start this project

- Image preprocessing(all the techniques)
- Feature extraction in image(all the techniques)
- Model building(whether it is genuine or forged)

How to work on this project

- **Data Collection**: Gather a dataset of genuine and forged signatures.
- **Preprocessing**: Process the images for uniformity and quality.
- **Feature Extraction:** Extract meaningful patterns from the signature images.
- **Model Training**: Train a classifier to distinguish between genuine and forged signatures.
- **Evaluation**: Test the model's accuracy and reliability.

Image Preprocessing

- Image preprocessing is the process of manipulating raw image data into a usable and meaningful format. It allows you to eliminate unwanted distortions and enhance specific qualities essential for computer vision applications.
- Preprocessing is a crucial first step to prepare your image data before feeding it into machine learning models.

There are different types of techniques

1. **Resizing:** Resizing images to a uniform size is important for machine learning algorithms to function properly.
2. **Grayscaling:** Converting color images to grayscale can simplify your image data and reduce computational needs for some algorithms.
3. **Noise reduction:** Smoothing, blurring, and filtering techniques can be applied to remove unwanted noise from images.
4. **Normalization:** Normalization adjusts the intensity values of pixels to a desired range, often between 0 to 1. This can improve the performance of machine learning models.
5. **Binarization:** Binarization converts grayscale images to black and white by thresholding.
6. **Contrast enhancement:** The contrast of images can be adjusted using **histogram equalization.**

What is Histogram equalization

There frequently arises the need to improve the contrast of the image. In such cases, we use an intensity transformation technique known as histogram equalization. Histogram equalization is the process of uniformly distributing the image histogram over the entire intensity axis by choosing a proper intensity transformation function. Hence, histogram equalization is an intensity transformation process.

Image Reizing

Image resizing changes the dimensions of an image. The challenge is to maintain image quality while altering its size.

a) Nearest Neighbor Interpolation:

- Simplest and fastest method
- Selects the pixel value of the nearest neighbor
- Results in a blocky image when enlarging

b) Bilinear Interpolation:

- Uses a weighted average of the 4 nearest pixel values
- Smoother results than nearest neighbor, but can cause some blurring

c) Bicubic Interpolation:

- Uses a weighted average of the 16 nearest pixel values
- Produces the smoothest results, especially for photographic images

CNN(Convolution Neural Networks)

What is CNN

A Convolutional Neural Network (CNN), also known as ConvNet, is a specialized type of deep learning algorithm mainly designed for tasks that necessitate object recognition, including image classification, detection, and segmentation. CNNs are employed in a variety of practical scenarios, such as autonomous vehicles, security camera systems, and others.

Why this is useful

CNNs are powerful because:

- **Feature Extraction:** They automatically learn **important features** (like edges, corners, textures) from images — no need for manual feature engineering.
- **Spatial Hierarchy:** Early layers capture **low-level patterns** (e.g., lines), while deeper layers learn **high-level features** (e.g., faces, objects).
- **Parameter Sharing:** A filter (kernel) is reused across the image, which reduces the number of parameters compared to a fully connected network — making it faster and less prone to overfitting.
- **Translation Invariance:** CNNs can recognize objects **regardless of their position** in the image.

When to use this CNN

These are some examples when to work on CNN

Use Case	Examples
Image Classification	Cats vs Dogs, Digit recognition (MNIST)
Object Detection	YOLO, SSD for real-time detection
Image Segmentation	Medical scans, satellite images
Video Analysis	Action recognition, surveillance
Document/Image OCR	Handwriting recognition, form reading
Audio Spectrograms	Audio classification, speech recognition

What are the components of CNN

- **Convolutional Layers:** These layers learn to detect various features in the images by applying different filters.
- **Pooling Layers:** These layers reduce the spatial dimensions of the feature maps, making the network more manageable and computationally efficient.
- **Fully Connected Layers:** These layers perform classification based on the features extracted by the convolutional and pooling layers.
- **Activation Functions**: Functions like ReLU and Sigmoid introduce non-linearity into the network, enabling it to learn complex patterns.
- **Loss Functions:** These functions measure the error between predicted and actual labels, guiding the optimization process.

How CNN works

- **Input Image**: CNN receives an input image which is preprocessed to ensure uniformity in size and format.
- **Convolutional Layers**: Filters are applied to the input image to extract features like edges, textures and shapes.
- **Pooling Layers**: The feature maps generated by the convolutional layers are downsampled to reduce dimensionality.
- **Fully Connected Layers**: The downsampled feature maps are passed through fully connected layers to produce the final output, such as a classification label.
- **Output**: The CNN outputs a prediction, such as the class of the image.

What is convolution layer

What this component Perform

The convolution operation involves a filter (or kernel) that slides over the input data, performing element-wise multiplications and summing the results to produce a feature map. This process allows the network to detect patterns such as edges, textures, and shapes in the input images.

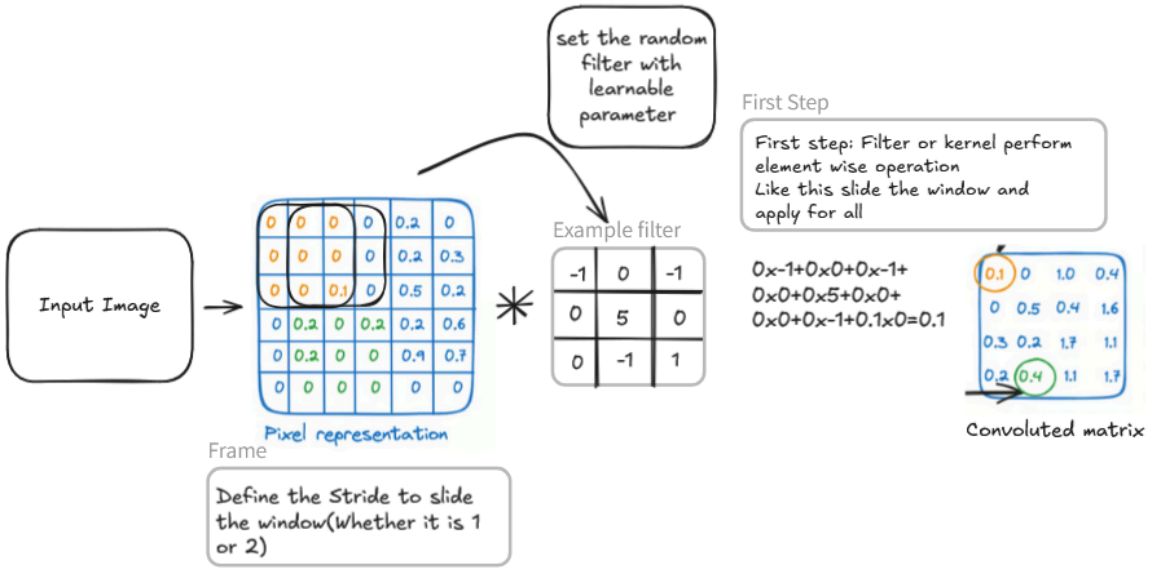
There are different types of components(Techniques)

1. **Filter or Kernel:** Filters are small, learnable matrices that extract specific features from the input data.
2. **Stride:** The stride determines how much the filter moves during the convolution operation. A stride of 1 means the filter moves one pixel at a time, while a stride of 2 means it moves two pixels at a time. Larger strides result in smaller output feature maps and faster computations.
3. **Padding**: Padding involves adding extra pixels around the input data to control the spatial dimensions of the output feature map. This are the common types of technique
    - Valid Padding: which adds no extra pixels, and

- same padding: which adds pixels to ensure the output feature map has the same dimensions as the input.

4. **Activation Function**: After the convolution operation, an activation function, typically the Rectified Linear Unit (ReLU), is applied to introduce non-linearity into the model. This helps the network learn complex patterns and relationships in the data.

How this Convolution works



1. **Initialize Filters:**
   - Randomly initialize a set of filters with learnable parameters.
2. **Convolve Filters with Input:**
   - Slide the filters across the width and height of the input data, computing the dot product between the filter and the input sub-region.
3. **Apply Activation Function:**
   - Apply a non-linear activation function to the convolved output to introduce non-linearity.
4. **Pooling (Optional):**
   - Often followed by a pooling layer (like max pooling) to reduce the spatial dimensions of the feature map and retain the most important information.

Activation Layer

A ReLU activation function is applied after each convolution operation. This function helps the network learn non-linear relationships between the features in the image, hence making the network more robust for identifying different patterns. It also helps to mitigate the vanishing gradient problems.

There are different types in this such as Sigmoid function, ReLU and Tanh

Mathematical representation

$$\text{Net Input} = \sum(\text{Weight} \times \text{Input}) + \text{Bias}$$

- Thus the activation function is an important part of an artificial neural network.
- They basically decide whether a neuron should be activated or not. Thus it bounds the value of the net input.
- The activation function is a non-linear transformation that we do over the input before sending it to the next layer of neurons or finalizing it as output.

Pooling Layer

- The goal of the pooling layer is to pull the most significant features from the convoluted matrix.
- **This is done by applying some aggregation operations, which reduce the dimension of the feature map (convoluted matrix),**
- hence reducing the spatial dimensions (width and height) of the input feature maps while retaining the most important information, memory used while training the network. Pooling is also relevant for mitigating overfitting.

Mathematical Representation

$$\left(\frac{n_h - f + 1}{s}\right) \times \left(\frac{n_w - f + 1}{s}\right) \times n_c$$

- nh → height of the feature map
- nw → width of the feature map
- nc → number of channels in the feature map
- ff → size of the pooling filter
- ss → stride length

Types of Pooling Layers

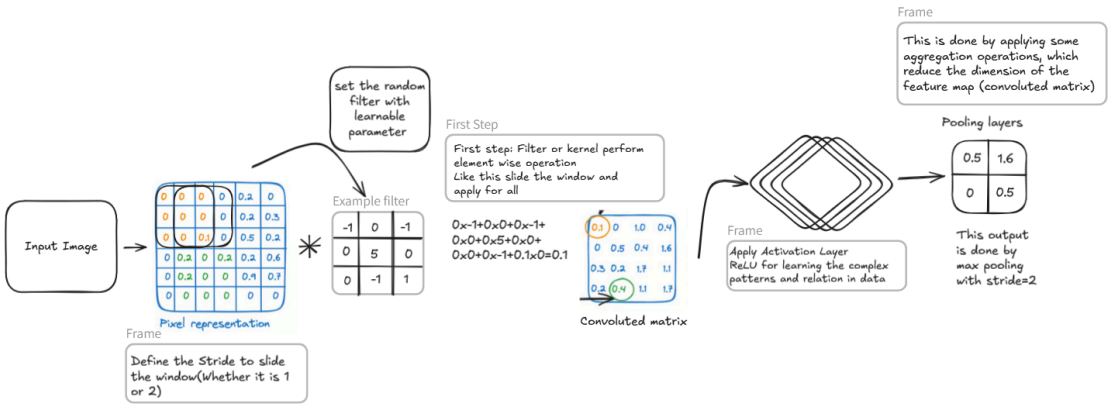
- Max pooling, which is the maximum value of the feature map
- Sum pooling corresponds to the sum of all the values of the feature map
- Average pooling is the average of all the values.

What does the max pooling:Max pooling layer preserves the most important features (edges, textures, etc.) and provides better performance in most cases.

What does the Average pooling:Average pooling provides a more generalized representation of the input. It is useful in the cases where preserving the overall context is important.

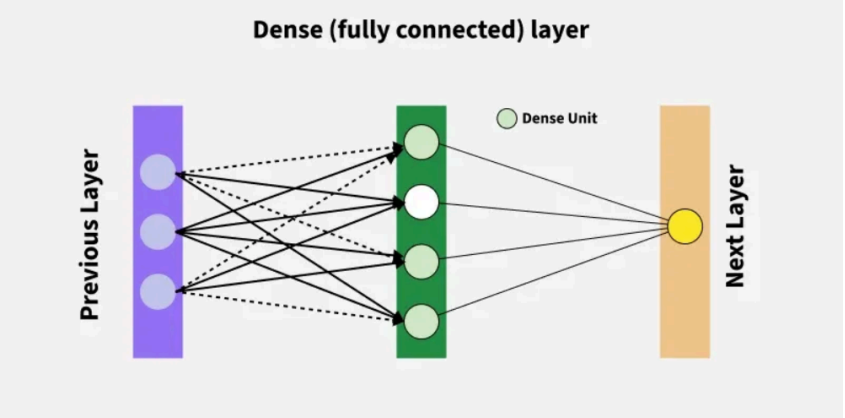
How Pooling works

1. Define the slide whether it (1,2 or 3) according to the case
2. Slide the window across the matrix with the defined the slide
3. Apply the pooling operator(Max,Average,Sum)
4. The result is a smaller feature map that retains the most important information.



Fully connected layer

- These layers are in the last layer of the convolutional neural network, and their inputs correspond to the flattened one-dimensional matrix generated by the last pooling layer. ReLU activation functions are applied to them for non-linearity.
- Finally, a softmax prediction layer is used to generate probability values for each of the possible output labels, and the final label predicted is the one with the highest probability score.



Components in Fully connected layer

- **Neurons**: Basic units that receive inputs from all neurons of the previous layer and send outputs to all neurons of the subsequent layer.
- **Weights**: Each connection between neurons has an associated weight indicating the strength and influence of one neuron on another.
- **Biases**: A bias term for each neuron helps adjust the output along with the weighted sum of inputs.
- **Activation Function**: Functions like ReLU, Sigmoid or Tanh introduce non-linearity to the model helping it to learn complex patterns and behaviors.

How to Train the CNN model

1. **Data Preparation:** The training images are preprocessed to ensure that they are all in the same format and size.
2. **Loss Function:** A loss function is used to measure how well the CNN is performing on the training data. The loss function is typically calculated by taking the difference between the predicted labels and the actual labels of the training images.
3. **Optimizer:** An optimizer is used to update the weights of the CNN in order to minimize the loss function.
4. **Backpropagation:** Backpropagation is a technique used to calculate the gradients of the loss function with respect to the weights of the CNN. The gradients are then used to update the weights of the CNN using the optimizer.

What happen the data is overfitted

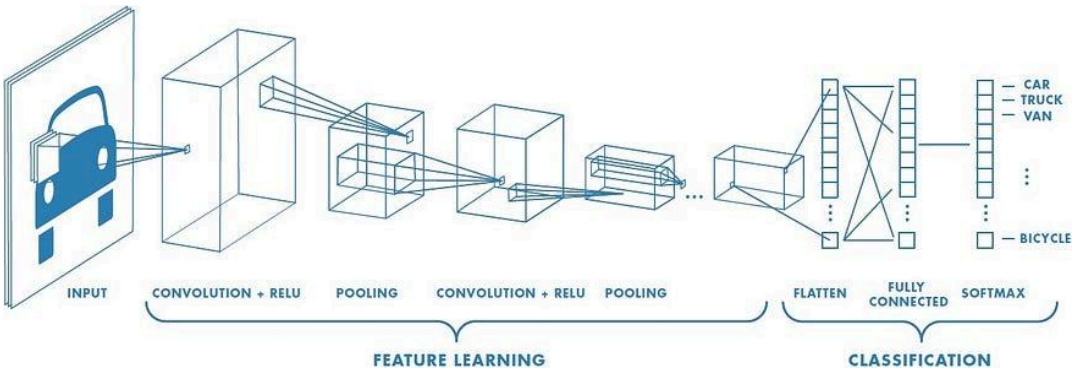
Overfitting is a common challenge in machine learning models and CNN deep learning projects. It happens when the model learns the training data too well ("learning by heart"), including its noise and outliers. Such a learning leads to a model that performs well on the training data but badly on new, unseen data.There are some techniques to use it

- **Dropout:** This consists of randomly dropping some neurons during the training process, which forces the remaining neurons to learn new features from the input data.
- **Batch normalization:** The overfitting is reduced at some extent by normalizing the input layer by adjusting and scaling the activations. This approach is also used to speed up and stabilize the training process.
- **Pooling Layers:** This can be used to reduce the spatial dimensions of the input image to provide the model with an abstracted form of representation, hence reducing the chance of overfitting.
- **Early stopping:** This consists of consistently monitoring the model's performance on validation data during the training process and stopping the training whenever the validation error does not improve anymore.
- **Data augmentation:** This is the process of artificially increasing the size and diversity of the training dataset by applying random transformations like rotation, scaling, flipping, or cropping to the input images.

How to evaluate the model

- **Accuracy:** Accuracy is the percentage of test images that the CNN correctly classifies.
- **Precision:** Precision is the percentage of test images that the CNN predicts as a particular class and that are actually of that class.
- **Recall:** Recall is the percentage of test images that are of a particular class and that the CNN predicts as that class.
- **F1 Score:** The F1 Score is a harmonic mean of precision and recall. It is a good metric for evaluating the performance of a CNN on classes that are imbalanced.

Basic structure pipeline of CNN



How to build the model practically(Flow of the project)

- Data collection
- Preprocessing
- Labelling: Whether it is Legal or Forged
- Train/Test
- Build CNN Model
- Model Training
- Model Evaluation
- Validation

How to build the CNN(Practically)

1.Define the Convolution layer

```
# Feature Learning Layers
  model.add(Conv2D(32,                # Number of filters/Kernels
                   (3,3),             # Size of kernels (3x3 matrix)
                   strides = 1,       # Step size for sliding the kernel across the input (1 pixel at a time).
                   padding = 'same',    # 'Same' ensures that the output feature map has the same dimensions as the input by padding zeros around the input.
```

```
            input_shape = (256,256,3)  # Input image shape
))
```

2.Define the Subsequent layer(Core of the Model)

```
    model.add(Activation('relu'))# Activation function
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size = (2,2), padding = 'same'))
    model.add(Dropout(0.2))
```

- This block represents a common pattern for a **convolutional block** within a CNN. It's repeated multiple times with increasing filter counts, which is characteristic of deep CNNs.

3. Repeat the Process

```
model.add(Conv2D(64, (5,5), padding = 'same'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2,2), padding = 'same'))
model.add(Dropout(0.2))
```

What it does

- **Convolutional Layer:** This is the core building block of a CNN.
- **64:** The number of filters (or kernels) in this layer. Each filter learns to detect a specific feature (e.g., edge, texture, pattern).
- **(5,5):** The size of the convolutional kernel (filter). A 5x5 filter can capture larger patterns than, say, a 3x3 filter.
- **padding = 'same':** Ensures that the output feature map has the same spatial dimensions (height and width) as the input feature map. By adding zero-padding

4. Next after 5x5 next is 3x3 (common practice while practicing CNN)

```
    model.add(Conv2D(128, (3,3), padding = 'same'))
    model.add(Activation('relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size = (2,2), padding = 'same'))
    model.add(Dropout(0.3))
```

- **Conv2D(128, (3,3), padding = 'same'):** The number of filters doubles to 128, and the kernel size changes to (3,3). Smaller kernels are often used in deeper layers to learn more intricate details from the features extracted by previous layers.

- Dropout: slightly higher risk of overfitting in this part of the network, or a desire for stronger regularization.

Repeat the layers(as per the requirement)

5. Flatten the image

```
# Flattening tensors
model.add(Flatten())
```

The Flatten() layer is essential for transforming the multi-dimensional feature maps generated by the convolutional and pooling layers into a format (a 1D vector) that can be processed by the subsequent fully connected (Dense) layers, which are typically used for the final classification or regression tasks in a CNN.

6. Fully connected layer and output layer

This both part of the pipeline is known as Decision-making for CNN

```
# Fully-Connected Layers
model.add(Dense(2048))
model.add(Activation('relu'))
model.add(Dropout(0.5))
```

In this part of the code

- **Dense:** Is the fully connected layer: This is a standard neural network layer where every neuron in this layer is connected to every neuron in the preceding layer.
- **2048**: This number specifies the **number of neurons (or units)** in this dense layer. This is a hyperparameter you choose
- **Activation Function (relu):** As seen in the convolutional blocks, ReLU (Rectified Linear Unit) is applied to the output of the preceding Dense layer.
- **Dropout Layer:** This is a regularization technique.
- **0.5:** This means during training, **50% of the neurons** in the preceding Dense layer (the one with 2048 units) will be randomly set to zero for each training batch.

7. Output layer(Classification)

In this part of the pipeline it does the classification whether it is genuine or forged signature for the project

```
# Output Layer
model.add(Dense(3, activation = 'softmax')) # Classification layer
```

- This is the final layer of your neural network, responsible for producing the model's prediction.
- The output Dense layer with softmax activation produces the probability distribution over your 3 defined classes (e.g., Genuine, Skilled Forgery, Random Forgery), giving you the model's prediction.

Final Full Pipeline of the CNN

```
# Initiating model on GPU
with strategy.scope():
    model = Sequential()

    model.add(augmentation) # Adding data augmentation pipeline to the model

    # Feature Learning Layers
    model.add(Conv2D(32,                  # Number of filters/Kernels
                     (3,3),               # Size of kernels (3x3 matrix)
                     strides = 1,         # Step size for sliding the kernel acr
                     padding = 'same',    # 'Same' ensures that the output featu
                    input_shape = (256,256,3) # Input image shape
                    ))
    model.add(Activation('relu'))# Activation function
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size = (2,2), padding = 'same'))
    model.add(Dropout(0.2))

    model.add(Conv2D(64, (5,5), padding = 'same'))
    model.add(Activation('relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size = (2,2), padding = 'same'))
    model.add(Dropout(0.2))

    model.add(Conv2D(128, (3,3), padding = 'same'))
    model.add(Activation('relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size = (2,2), padding = 'same'))
    model.add(Dropout(0.3))

    model.add(Conv2D(256, (5,5), padding = 'same'))
    model.add(Activation('relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size = (2,2), padding = 'same'))
    model.add(Dropout(0.3))

    model.add(Conv2D(512, (3,3), padding = 'same'))
    model.add(Activation('relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size = (2,2), padding = 'same'))
    model.add(Dropout(0.3))

    # Flattening tensors
    model.add(Flatten())

    # Fully-Connected Layers
    model.add(Dense(2048))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))

    # Output Layer
    model.add(Dense(3, activation = 'softmax')) # Classification layer
```