Project Name: End-To-End ML with Deployment Cotton Plant Disease Prediction (FileUse) – Convolution Neural Network (CNN) – Classification

Table of Contents

```
#import libraries
import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
import matplotlib.pyplot as plt

keras.__version__

# Uncomment below this two lines if performing in Google Colab. - IT IS IMPORTANT STEP.
#from google.colab import drive
#drive.mount('/content/drive')

# I have uploaded Datasets Folder on Google Drive from my laptop in My DL Project - you also have to upload it on your drive.
train_data_path = "/content/drive/My Drive/My DL Project/cotton plant disease prediction/Datasets/train"
validation_data_path = "/content/drive/My Drive/My DL Project/cotton plant disease prediction/Datasets/val"

def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20, 20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.show()

# this is the augmentation configuration we will use for training
# It generate more images using below parameters
training_datagen = ImageDataGenerator(rescale=1./255,
                                      rotation_range=40,
                                      width_shift_range=0.2,
                                      height_shift_range=0.2,
                                      shear_range=0.2,
                                      zoom_range=0.2,
                                      horizontal_flip=True,
                                      fill_mode='nearest')
```

```
# this is a generator that will read pictures found in
# at train_data_path, and indefinitely generate
# batches of augmented image data
training_data = training_datagen.flow_from_directory(train_data_path, # this is the target directory
                                      target_size=(150, 150), # all images will be resized to 150x150
                                      batch_size=32,
                                      class_mode='binary')  # since we use binary_crossentropy loss, we need binary labels

training_data.class_indices

# this is the augmentation configuration we will use for validation:
# only rescaling
valid_datagen = ImageDataGenerator(rescale=1./255)

# this is a similar generator, for validation data
valid_data = valid_datagen.flow_from_directory(validation_data_path,
                                    target_size=(150,150),
                                    batch_size=32,
                                    class_mode='binary')

images = [training_data[0][0][0] for i in range(5)]
plotImages(images)

# save best model using val accuracy
model_path = '/content/drive/My Drive/My DL Project/cotton plant disease prediction/v3_red_cott_dis.h5'
checkpoint = ModelCheckpoint(model_path, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]
```

```
#Building cnn model
cnn_model = keras.models.Sequential([
                                    keras.layers.Conv2D(filters=32, kernel_size=3, input_shape=[150, 150, 3]),
                                    keras.layers.MaxPooling2D(pool_size=(2,2)),
                                    keras.layers.Conv2D(filters=64, kernel_size=3),
                                    keras.layers.MaxPooling2D(pool_size=(2,2)),
                                    keras.layers.Conv2D(filters=128, kernel_size=3),
                                    keras.layers.MaxPooling2D(pool_size=(2,2)),
                                    keras.layers.Conv2D(filters=256, kernel_size=3),
                                    keras.layers.MaxPooling2D(pool_size=(2,2)),

                                    keras.layers.Dropout(0.5),
                                    keras.layers.Flatten(), # neural network beulding
                                    keras.layers.Dense(units=128, activation='relu'), # input layers
                                    keras.layers.Dropout(0.1),
                                    keras.layers.Dense(units=256, activation='relu'),
                                    keras.layers.Dropout(0.25),
                                    keras.layers.Dense(units=4, activation='softmax') # output layer
])


# compile cnn model
cnn_model.compile(optimizer = Adam(lr=0.0001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# train cnn model
history = cnn_model.fit(training_data,
                        epochs=500,
                        verbose=1,
                        validation_data= valid_data,
                        callbacks=callbacks_list)
```

```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

history.history
```

Live Demo Link

Deployment on Heroku:

Abstract

The purpose of this report will be to use the Cotton Disease Data to classify type of plant or leaf into a possible diseased cotton plant/leaf or fresh cotton plant/leaf.
This can be used to gain insight into how and why plant/leaf is curable or cure is not required. This can also be used as a model to gain a marketing advantage, by advertisement targeting those who

are more likely to necessary future life skill of hydro-farming or basic planting in apartments locally especially after corona pandemic.
Cotton Disease Prediction is an image classification problem, where using the images model will classify what kind of problems are in the images.

This is diving into Cotton Disease Image Classification through Machine Learning Concept.
End to End Project means that it is step by step process, starts with data collection, Resizing Data, Rescaling Data, Data Preparation which includes standardizing and transforming then selecting, training and saving DL Models, Cross-validation and Hyper-Parameter Tuning and developing web service then Deployment for end users to use it anytime and anywhere.
This repository contains the code for Cotton Disease Image Classification using python's various libraries.
It used matplotlib and keras libraries.
These libraries help to perform individually one particular functionality.
Matplotlib is a plotting library.
Keras is built in Python which makes it way more user-friendly than TensorFlow.
These python libraries raised knowledge in discovering these libraries with practical use of it.
It leads to growth in my DL repository.
These above screenshots and video in Video_File Folder will help you to understand flow of output.

Motivation

As I have visited my hometown many times during vacations, I know Farmer can't solve Farm's complex and even small problems due to lack of perfect education. So as AI enthusiastic I decided to solve this problem using the latest technology like AI with Deep Learning Concept.

Acknowledgement

Dataset Available: https://www.kaggle.com/janmejaybhoi/cotton-disease-dataset
OR
https://drive.google.com/drive/folders/1vdr9CC9ChYVW2iXp6PlfyMOGD-4Um1ue

The Data

After Downloading the dataset unzip it and place it under the Datasets Folder, all the path in the Colab notebook is according to my relative path so it needs to be changed accordingly.
An end-to-end application which predicts whether the cotton plant belongs to the following set of classes.

- Diseased Cotton Leaf.
- Diseased Cotton Plant.
- Fresh Cotton Leaf.
- Fresh Cotton Plant.

I just took baby step and start to collect lots of images of cotton crop plants from my farm. To collect accurate data, we need expertise in that domain and as a farmer it helps me a lot.

- Math behind the metrics

When we unlock the phone with our face or automatically retouch photos before posting them on social media. Convolutional Neural Networks are possibly the most crucial building blocks behind these huge successes. This time we are going to broaden our understanding of how neural networks work.

Step1: Neural Net

These are networking whose neurons are divided into groups forming successive layers. Each such unit is connected to every single neuron from the neighbouring layers. An example of such an architecture is shown in the figure below.

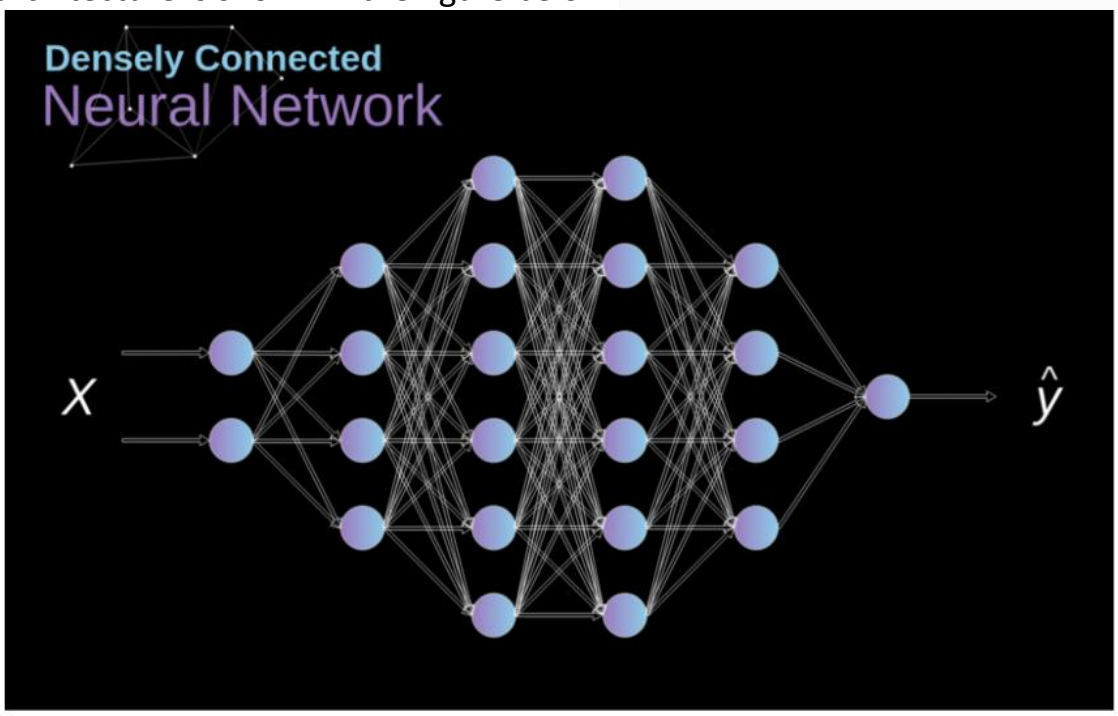


Figure 1. Densely connected neural network architecture

This approach works well when we solve classification problem based on a limited set of defined features. However, the situation becomes more complicated when working with photos.

Step 2: Digital Photo Data Structure

Digital images are stored as actually huge matrices of numbers. Each such number corresponds to the brightness of a single pixel. In the RGB model, the colour image is actually composed of three such matrices corresponding to three colour channels — red, green and blue. In black-and-white images we only need one matrix. Each of these matrices stores values from 0 to 255. This range is a compromise between the efficacy of storing information about the image (256 values fit perfectly in 1 byte) and the sensitivity of the human eye (we distinguish a limited number of shades of the same colour).

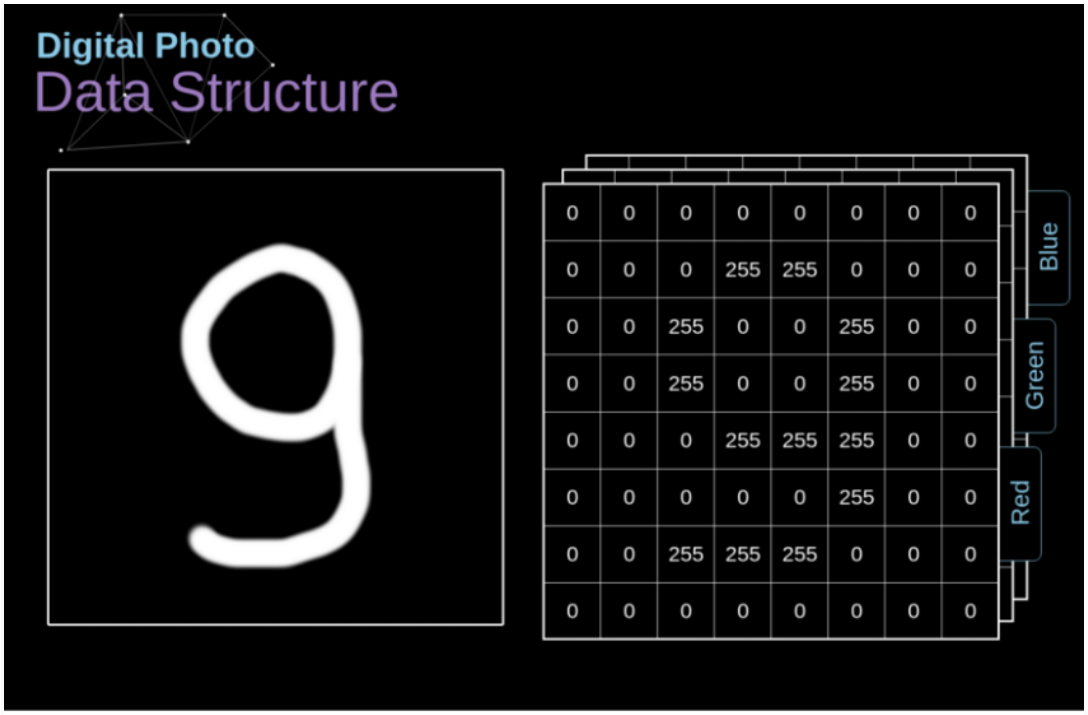


Figure 2. Data structure behind digital images

Step 3: Convolution

Kernel convolution is not only used in CNNs, but is also a key element of many other Computer Vision algorithms. It is a process where we take a small matrix of numbers (called kernel or filter), we pass it over our image and transform it based on the values from filter**.** Subsequent feature map values are calculated according to the following formula, where the input image is denoted by *f* and our kernel by *h*. The indexes of rows and columns of the result matrix are marked with *m* and *n* respectively.

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k] f[m-j, n-k]$$

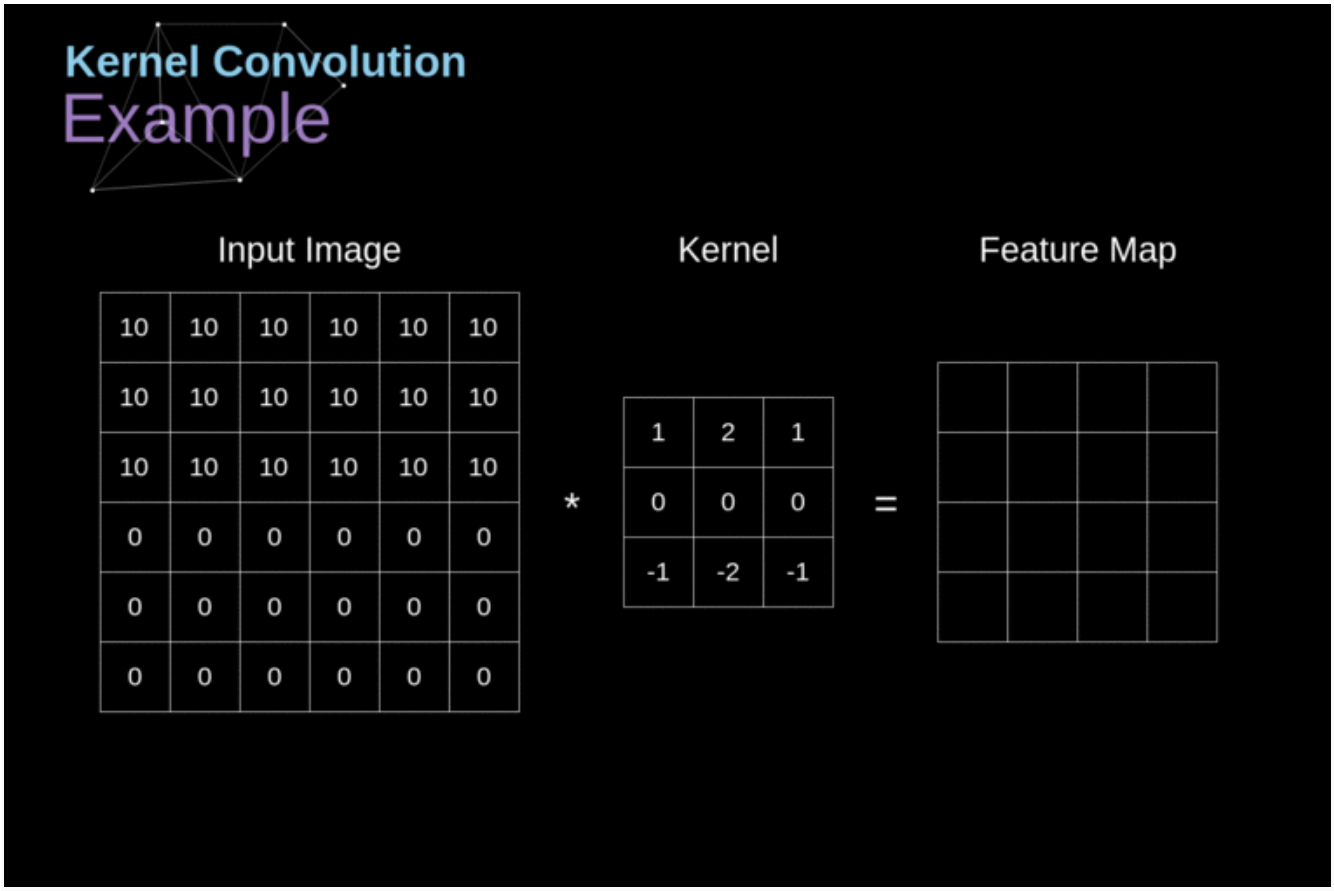


Figure 3. Kernel convolution example

After placing our filter over a selected pixel, we take each value from kernel and multiply them in pairs with corresponding values from the image. Finally, we sum up everything and put the result in the right place in the output feature map. Above we can see how such an operation looks like in micro scale, but what is even more interesting, is what we can achieve by performing it on a full image. Figure 4 shows the results of the convolution with several different filters.

Figure 4. Finding edges with kernel convolution

Step 4: Valid and Same Convolution

we have seen in Figure 3, when we perform convolution over the 6x6 image with a 3x3 kernel, we get a 4x4 feature map. This is because there are only 16 unique positions where we can place our filter inside this picture. Since our image shrinks every time, we perform convolution, we can do it only a limited number of times, before our image disappears completely. What's more, if we look at how our kernel moves through the image, we see that the impact of the pixels located on the outskirts is much smaller than those in the center of image. This way we lose some of the information contained in the picture. Below you can see how the position of the pixel changes its influence on the feature map.

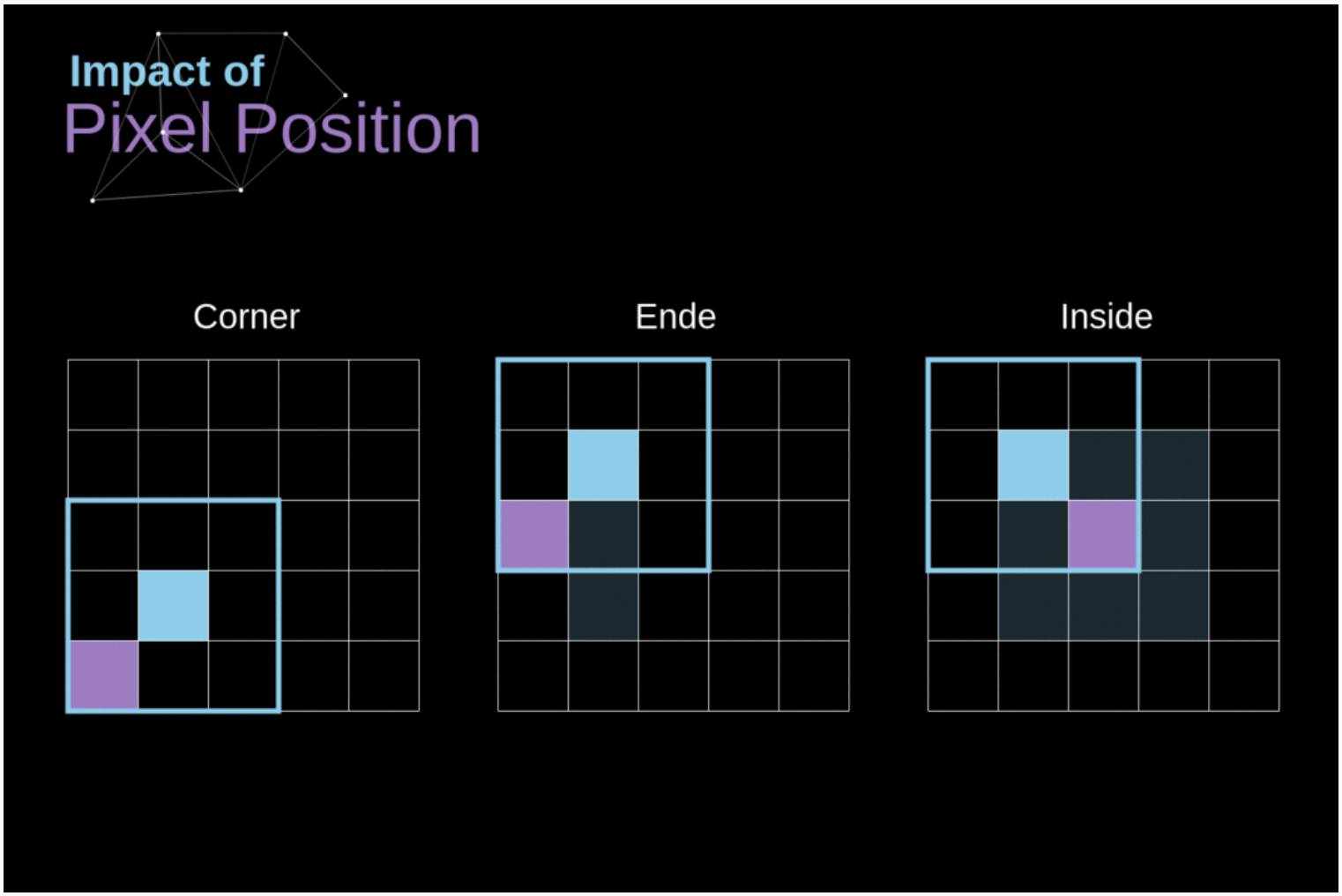


Figure 5. Impact of pixel position

To solve both of these problems we can pad our image with an additional border. For example, if we use 1px padding, we increase the size of our photo to 8x8, so that output of the convolution with the 3x3 filter will be 6x6. Usually in practice we fill in additional padding with zeroes. Depending on whether we use padding or not, we are dealing with two types of convolution — Valid and Same. Naming is quite unfortunate, so for the sake of clarity: Valid — means that we use the original image, Same — we use the border around it**,** so that the images at the input and output are the same size. In the second case, the padding width, should meet the following equation, where *p* is padding and *f* is the filter dimension (usually odd).

$$p = \frac{f-1}{2}$$

Step 5: Strided Convolution

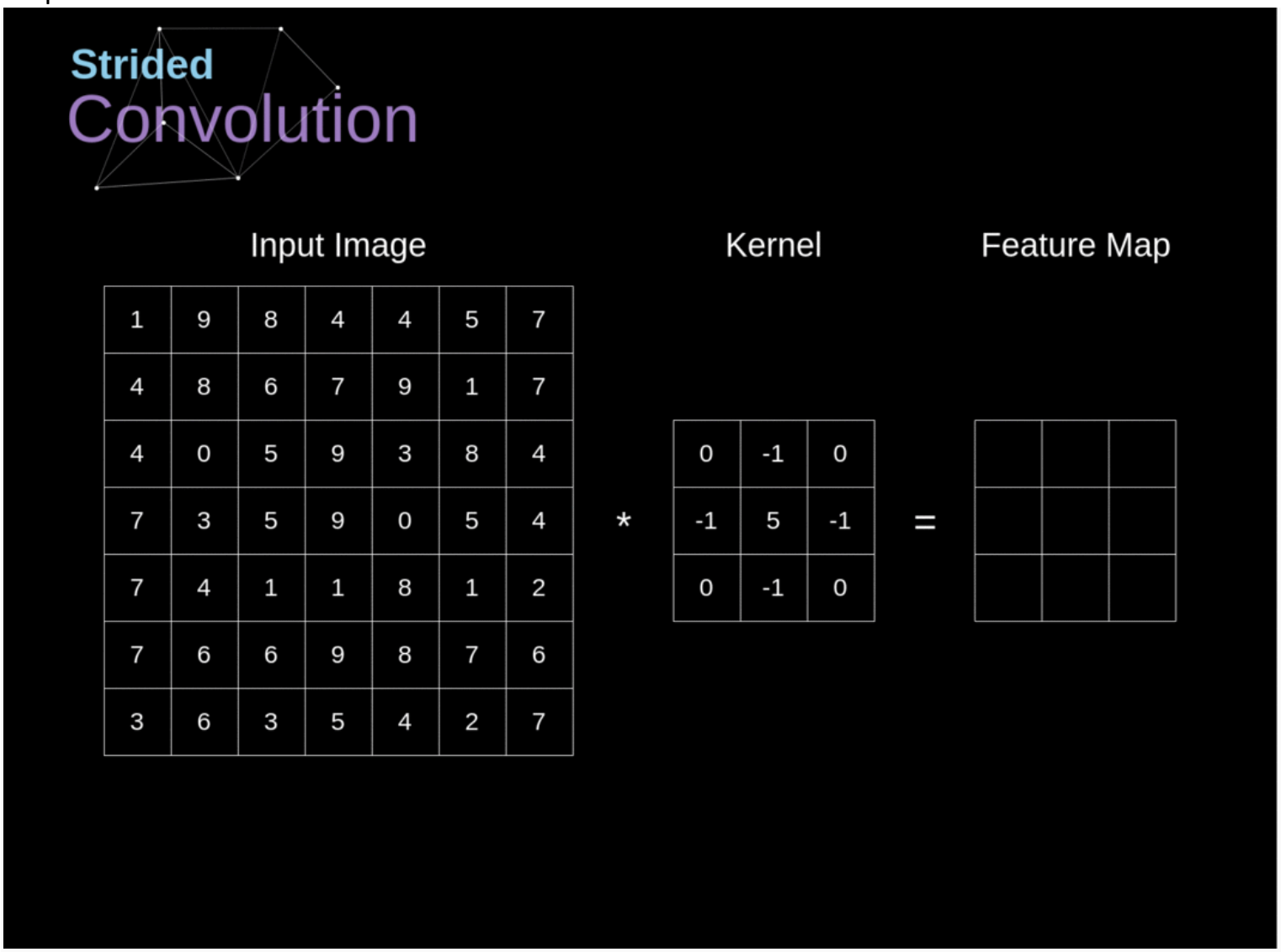


Figure 6. Example of strided convolution

In Figure 6, we can see how the convolution looks like if we use larger stride. When designing our CNN architecture, we can decide to increase the step if we want the receptive fields to overlap less or if we want smaller spatial dimensions of our feature map. The dimensions of the output matrix - taking into account padding and stride - can be calculated using the following formula.

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - f}{s} + 1 \right\rfloor$$

Step 6: The transition to the third dimension

Convolution over volume is a very important concept, which will allow us not only to work with color images, but even more importantly to apply multiple filters within a single layer. The first important rule is that the filter and the image you want to apply it to, must have the same number of channels. Basically, we proceed very much like in the example from Figure 3, nevertheless this time we multiply the pairs of values from the three-dimensional space. If we want use multiple filters on the same image, we carry out the convolution for each of them separately, stack the results one on top of the other and combine them into a whole. The dimensions of the received tensor (as our 3D matrix can be called) meet the following equation, in which: *n* — image size, *f* — filter size, *nc* — number of channels in the image, *p* —used padding, *s* — used stride, *nf* — number of filters.

$$[n, n, n_c] * [f, f, n_c] = \left[\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor, \left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor, n_f \right]$$

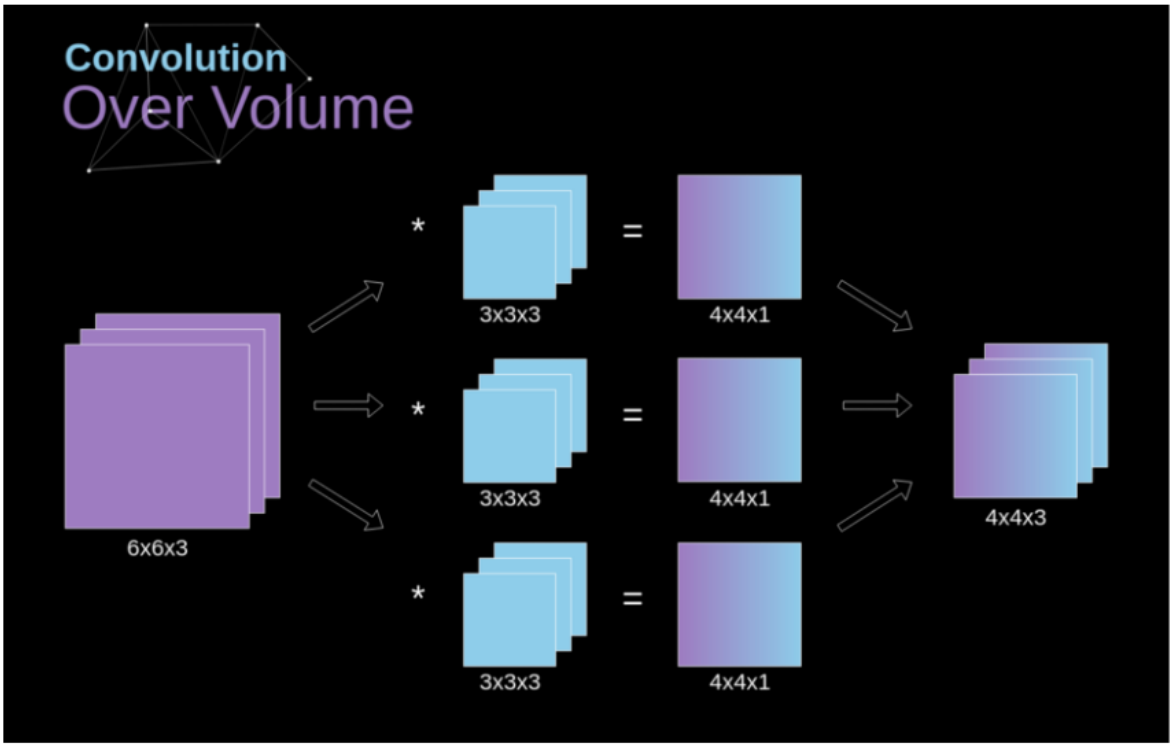


Figure 7. Convolution over volume

Step 7: Convolution Layers

Forward propagation consists of two steps. The first one is to calculate the intermediate value Z, which is obtained as a result of the convolution of the input data from the previous layer with W tensor (containing filters), and then adding bias b. The second is the application of a non-linear activation function to our intermediate value (our activation is denoted by g). Fans of matrix equations will find appropriate mathematical formulas below. By the way, on illustration below you can see a small visualization, describing the dimensions of tensors used in equation.

$$\mathbf{Z}^{[l]} = \mathbf{W}^{[l]} \cdot \mathbf{A}^{[l-1]} + \mathbf{b}^{[l]} \qquad \mathbf{A}^{[l]} = g^{[l]}(\mathbf{Z}^{[l]})$$

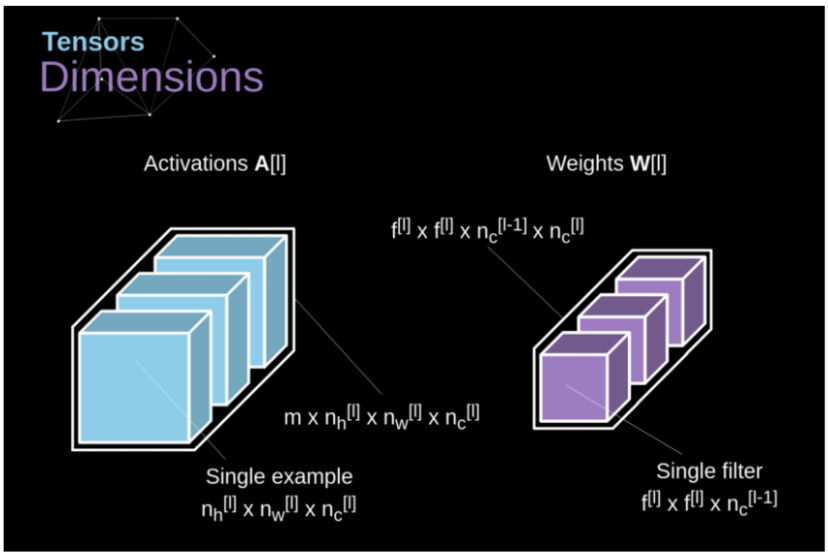


Figure 8. Tensors dimensions

Step 8: Connections Cutting and Parameters Sharing

Now that we understand what convolution is all about, let's consider how it allows us to optimize the calculations. On the Figure below, the 2D convolution has been visualized in a slightly different way — neurons marked with numbers 1–9 form the input layer that receives brightness of subsequent pixels, while units A-D denotes calculated feature map elements. Last but not least, I-IV are the subsequent values from kernel — these must be learned.

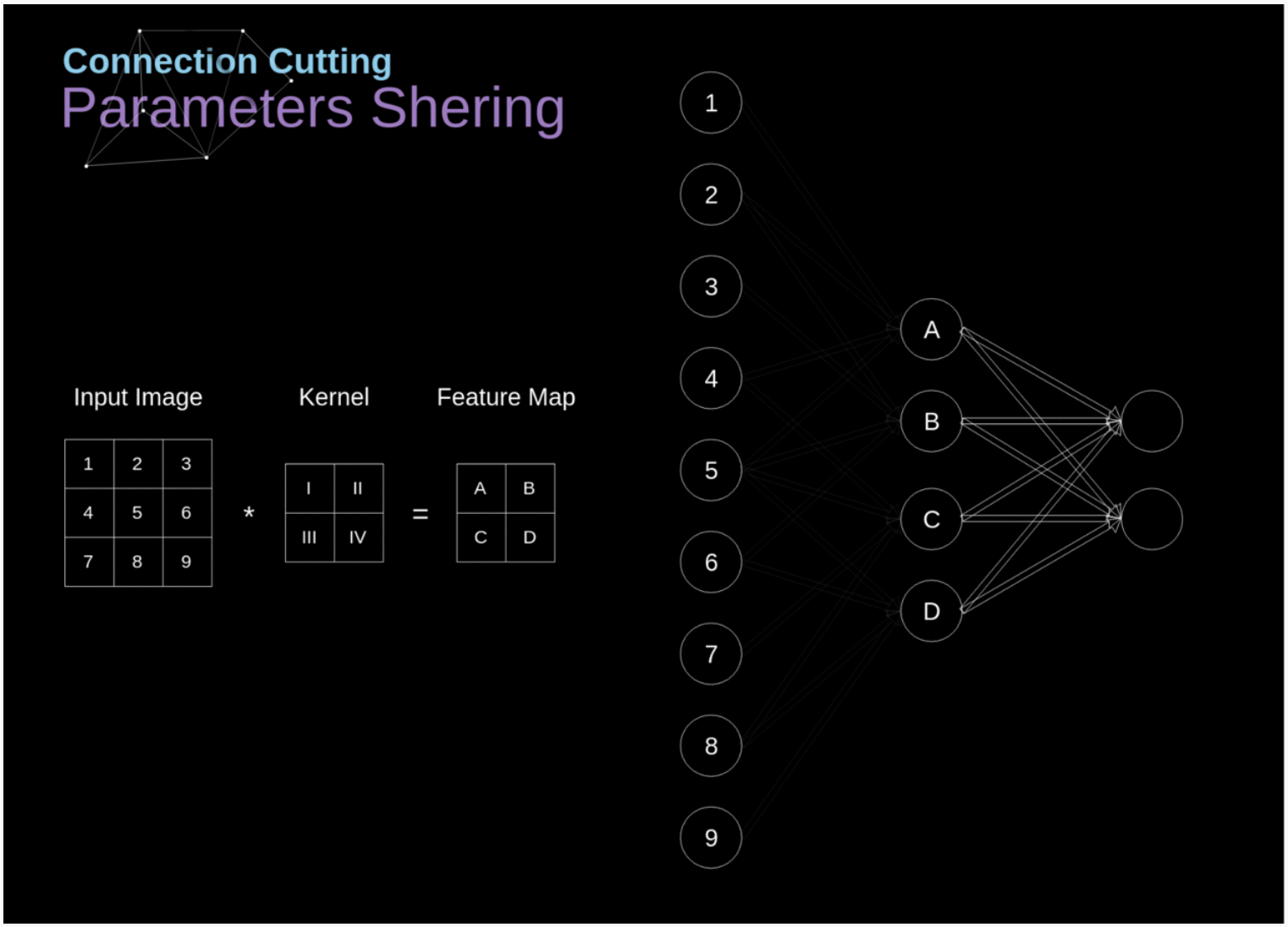


Figure 9. Connections cutting and parameters sharing

Now, let's focus on the two very important attributes of convolution layers. First of all, you can see that not all neurons in the two consecutive layers are connected to each other. For example, unit 1 only affects the value of A. Secondly, we see that some neurons share the same weights. Both of these properties mean that we have much less parameters to learn. By the way, it is worth noting that a single value from the filter affects every element of the feature map — it will be crucial in the context of backpropagation.

Step 9: Convolutional Layer Backpropagation

Our goal is to calculate derivatives and later use them to update the values of our parameters in a process called gradient descent. In our calculations we will use a chain rule — which I mentioned in previous articles. We want to assess the influence of the change in the parameters on the resulting features map, and subsequently on the final result. Before we start to go into the details, let us agree on the mathematical notation that we will use — in order to make my life easier, I will abandon the full notation of the partial derivative in favour of the shortened one visible below. But

remember, that when I use this notation, I will always mean the partial derivative of the cost function.

$$\mathbf{dA}^{[l]} = \frac{\partial L}{\partial \mathbf{A}^{[l]}} \quad \mathbf{dZ}^{[l]} = \frac{\partial L}{\partial \mathbf{Z}^{[l]}} \quad \mathbf{dW}^{[l]} = \frac{\partial L}{\partial \mathbf{W}^{[l]}} \quad \mathbf{db}^{[l]} = \frac{\partial L}{\partial \mathbf{b}^{[l]}}$$

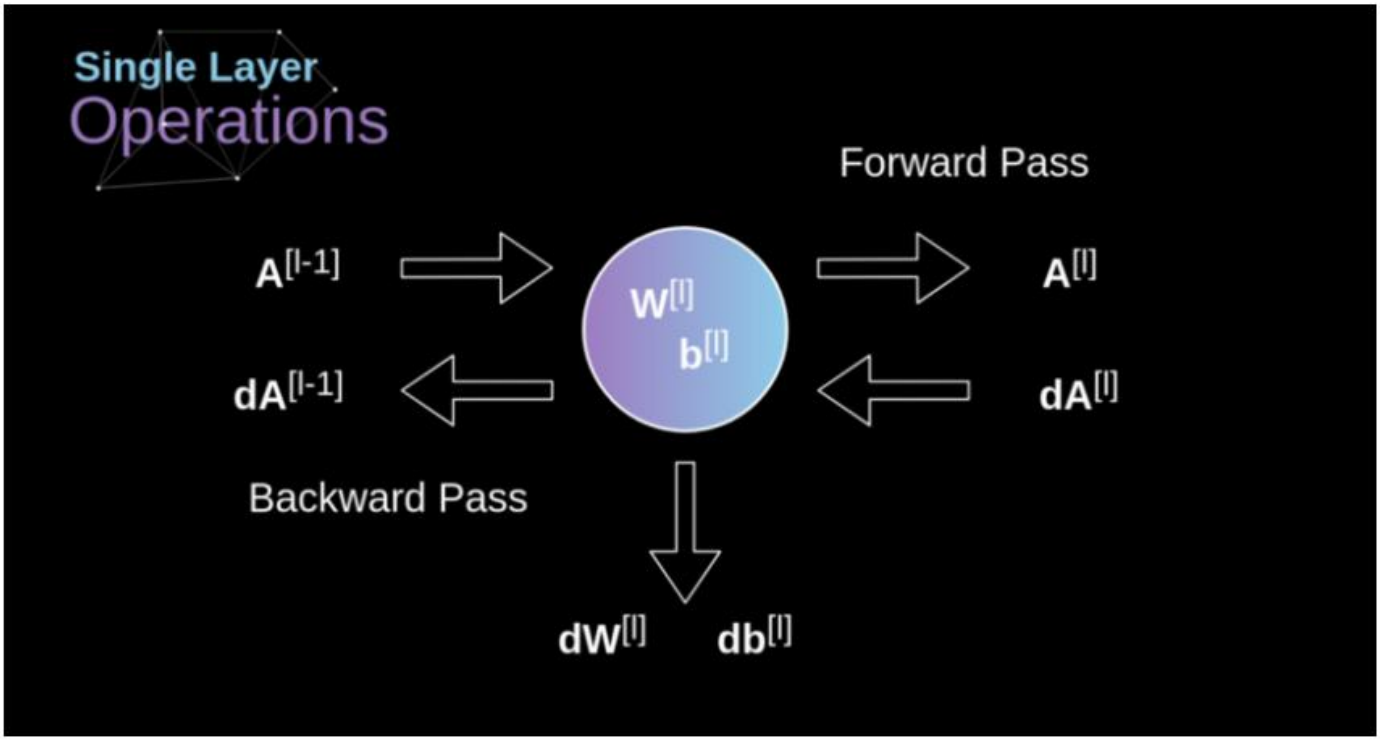


Figure 10. Input and output data for a single convolution layer in forward and backward propagation

Our task is to calculate **dW[l]** and **db[l] -** which are derivatives associated with parameters of current layer, as well as the value of **dA[l -1]** - which will be passed to the previous layer. As shown in Figure 10, we receive the **dA[l]** as the input. Of course, the dimensions of tensors **dW** and **W**, **db** and **b** as well as **dA** and **A** respectively are the same. The first step is to obtain the intermediate value **dZ[l]** by applying a derivative of our activation function to our input tensor. According to the chain rule, the result of this operation will be used later.

$$\mathbf{dZ}^{[l]} = \mathbf{dA}^{[l]} * g'(\mathbf{Z}^{[l]})$$

Now, we need to deal with backward propagation of the convolution itself, and in order to achieve this goal we will utilise a matrix operation called full convolution — which is visualised below. Note that during this process we use the kernel, which we previously rotated by 180 degrees. This operation can be described by the following formula, where the filter is denoted by **W**, and **dZ[m,n]** is a scalar that belongs to a partial derivative obtained from the previous layer.

$$\mathbf{dA}+ = \sum_{m=0}^{n_h} \sum_{n=0}^{n_w} \mathbf{W} \cdot \mathbf{dZ}[m, n]$$

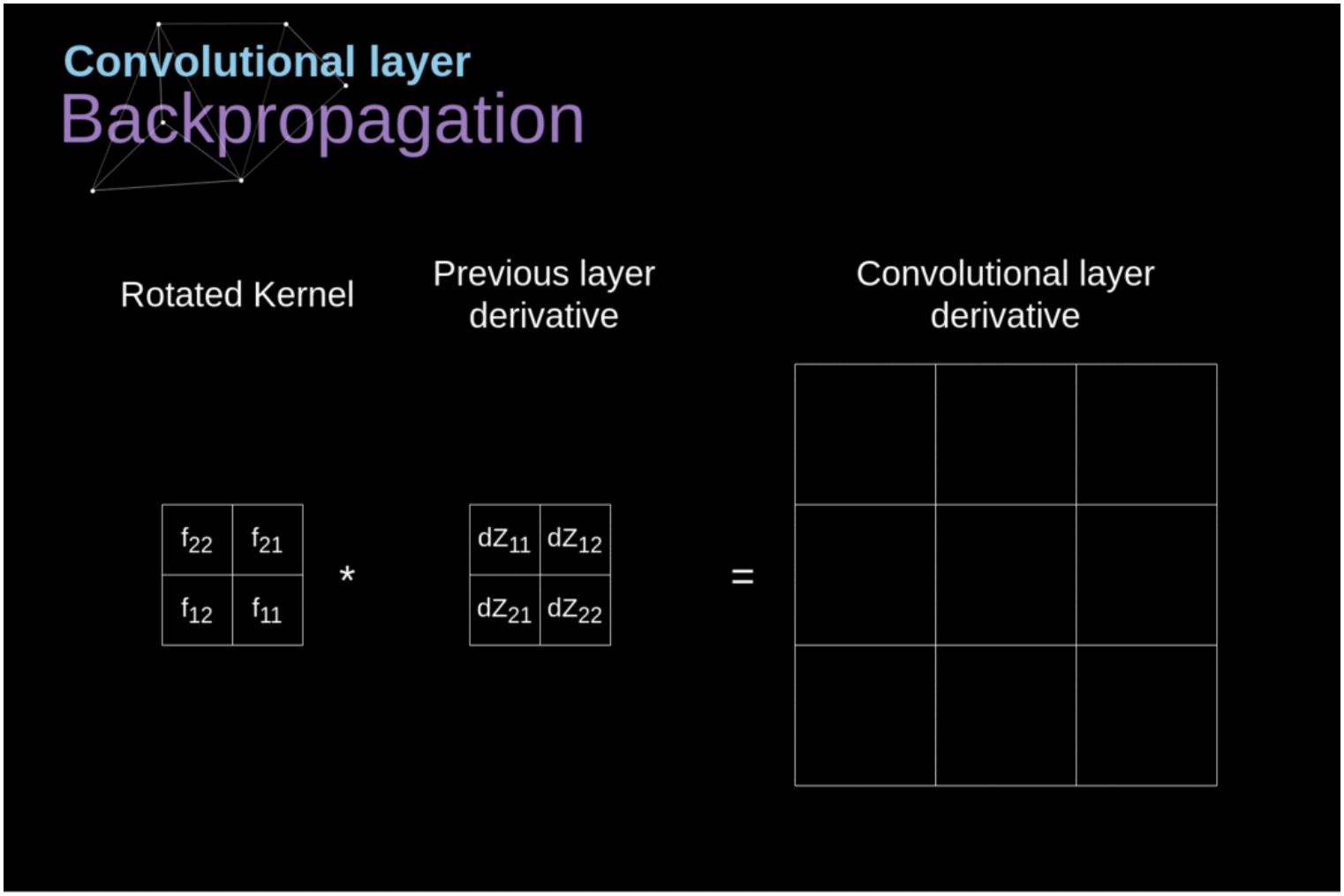
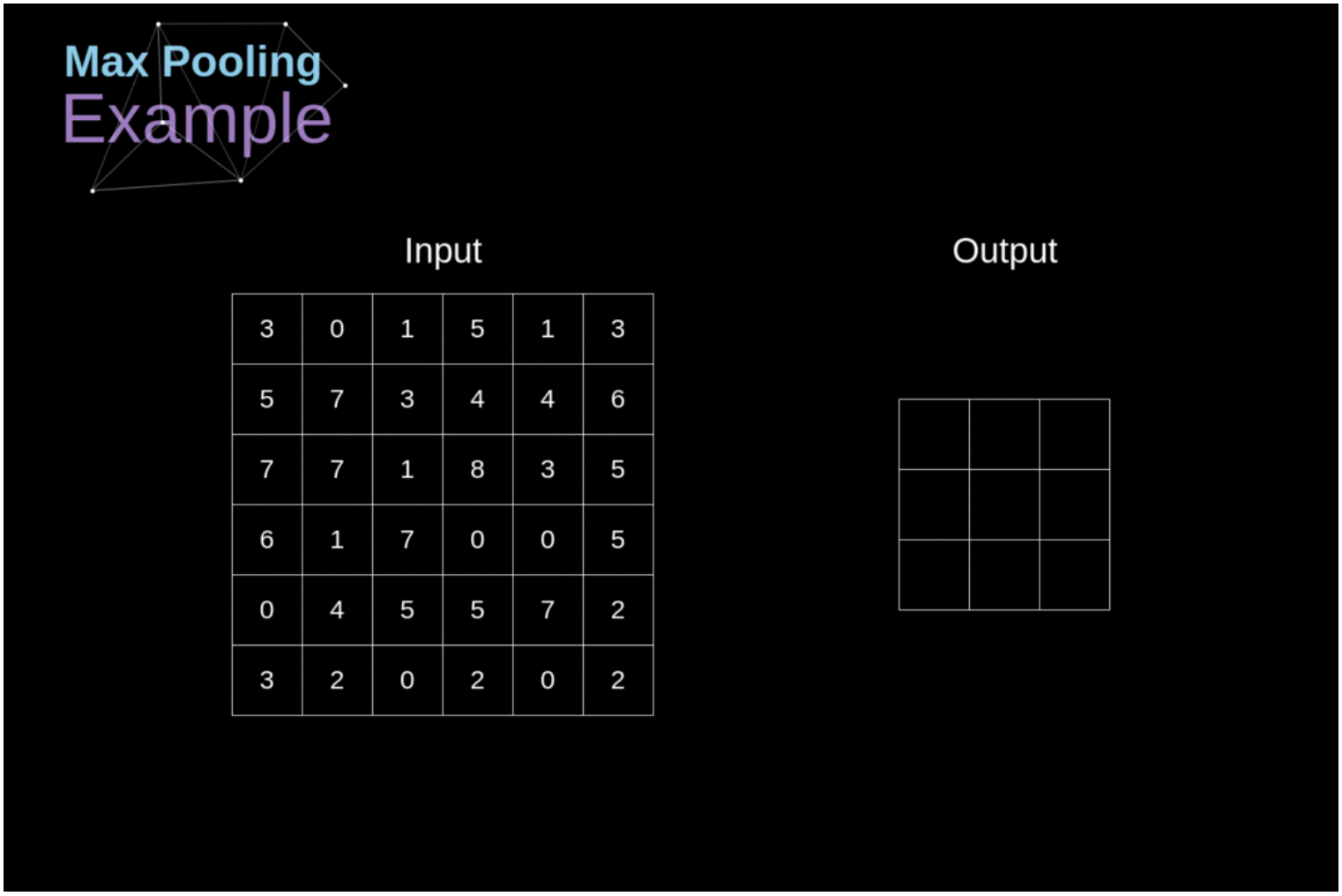


Figure 11. Full convolution

Step 10: Pooling Layers

They are used primarily to reduce the size of the tensor and speed up calculations. These layers are simple - we need to divide our image into different regions, and then perform some operation for each of those parts. For example, for the Max Pool Layer, we select a maximum value from each region and put it in the corresponding place in the output. As in the case of the convolution layer, we have two hyperparameters available — filter size and stride. Last but not least, if you are performing pooling for a multi-channel image, the pooling for each channel should be done separately.



Step 11: Pooling Layers Backpropagation

During back propagation, the gradient should not affect elements of the matrix that were not included in the forward pass. In practice, this is achieved by creating a mask that remembers the position of the values used in the first phase, which we can later utilize to transfer the gradients.

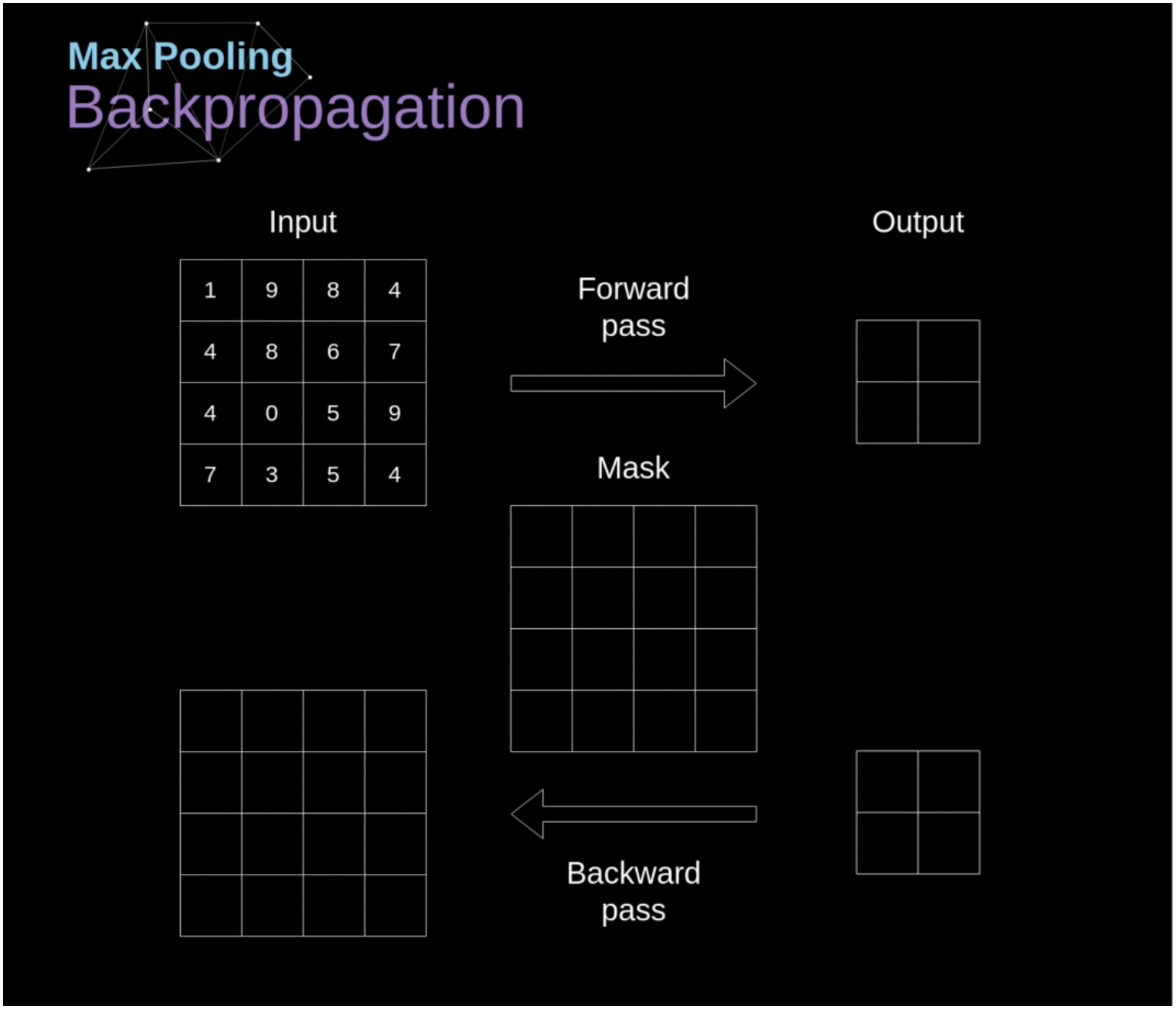
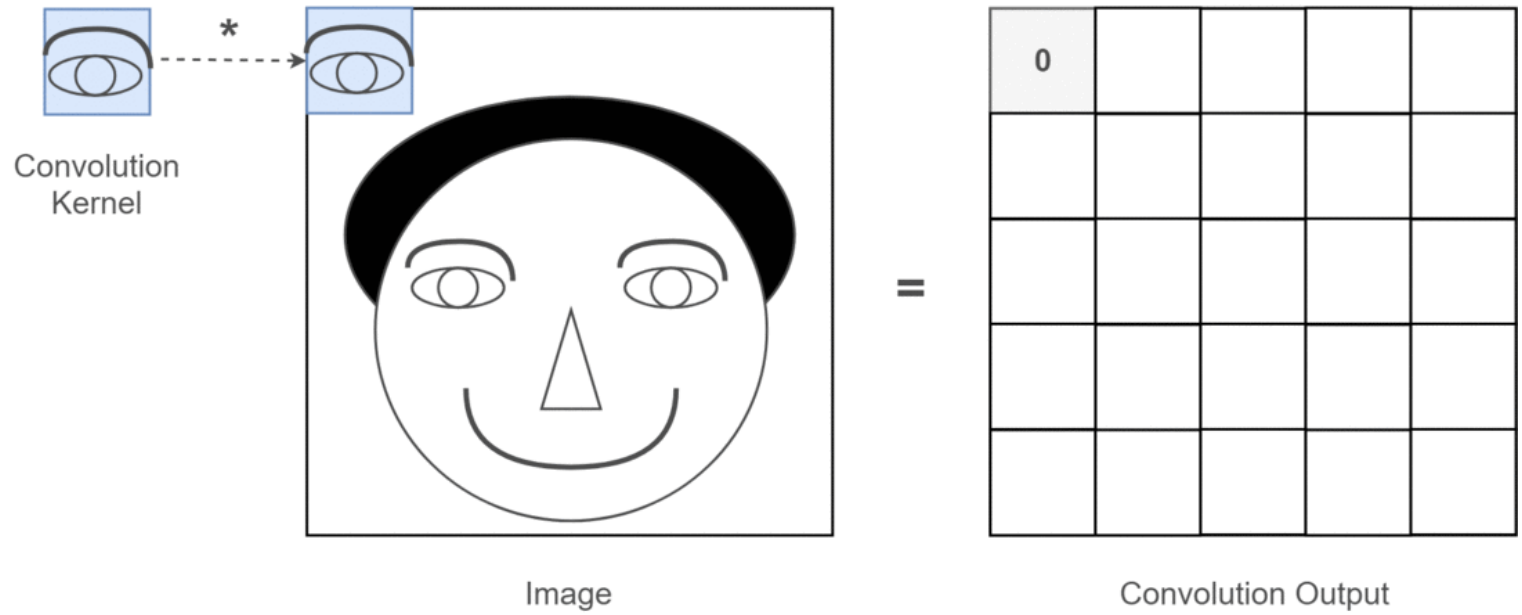


Figure 13. Max pooling backward pass

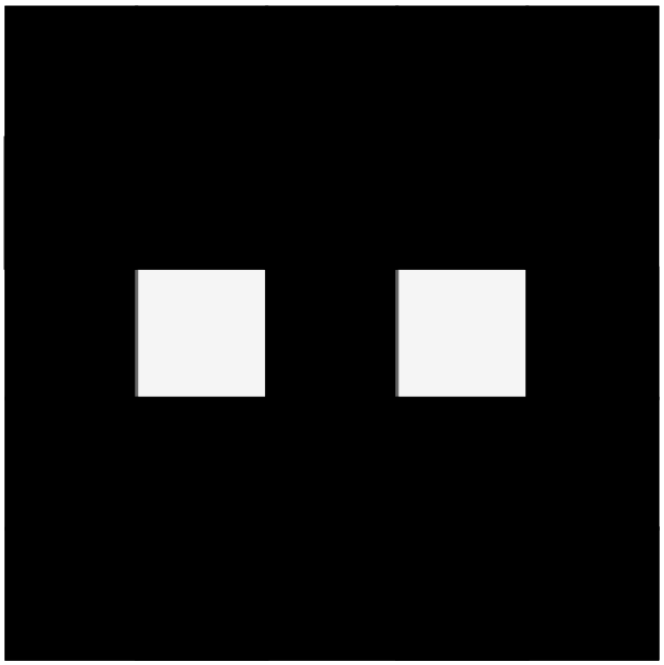
- Model Architecture Process Through Visualization

CNN through Visualization



0	0	0	0	0
0	0	0	0	0
0	100	0	100	0
0	0	0	0	0
0	0	0	0	0

Convolution Output



Convolution Output
(as an Image)

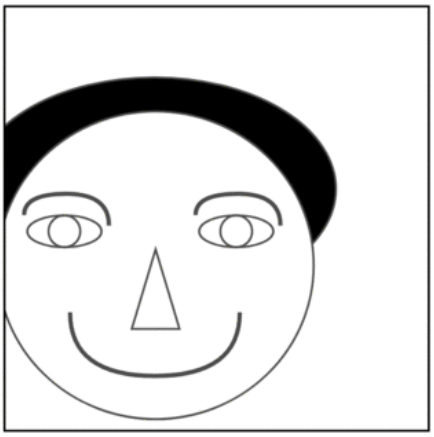
Pooling

Image

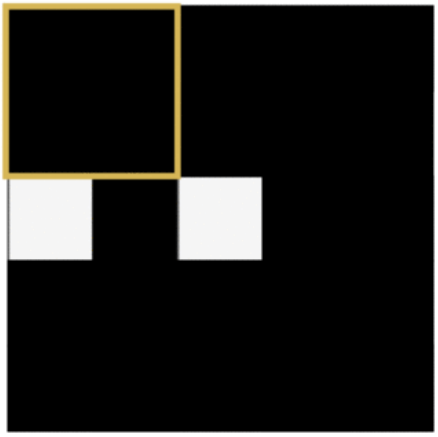
Convolution Output
(as an Image)

Pooling Output (as
an Image)

Same image as above but
translated on x-axis



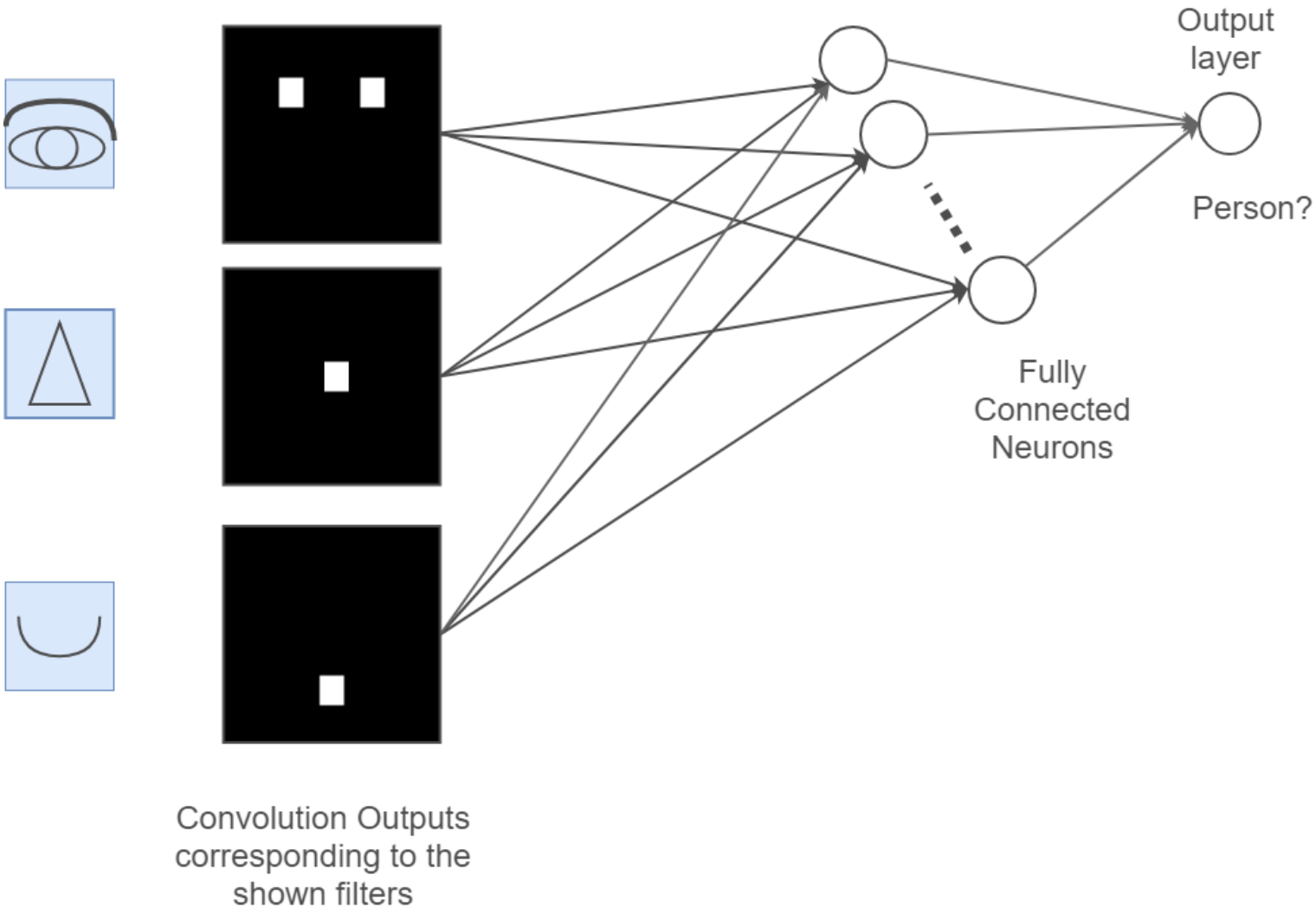
Image



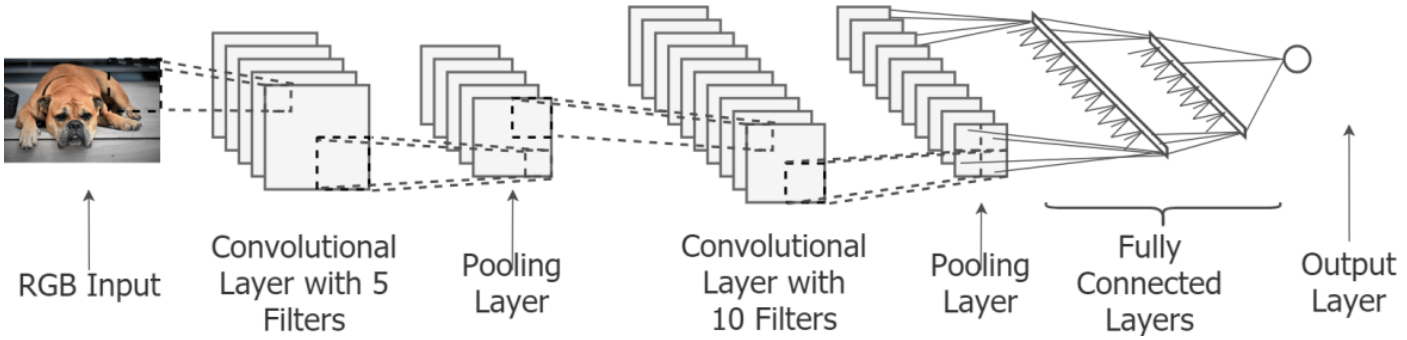
Convolution Output
(as an Image)

Pooling Output (as
an Image)

Fully Connected Layers



Result



- Quick Notes

Step 1: Keep Dataset Ready.
Step 2: Import Libraries.
Step 3: Load the Data.
Step 4: Performed Data Preprocessing and Data Augmentation to generate more images and batches of augmented data for Training_Data and Validation_Data.
Step 5: Visualize images generated to gain insights.
Step 6: Created a model with .h5 extension.
Step 7: Created Checkpoints.
Step 8: Built the CNN Model.
Step 9: Compiled the CNN Model.
Step 10: Trained the CNN Model on Training and Validation Data.
Step 11: Checked Accuracy through Visualization and then created Web App.

- The Model Analysis

Kept Dataset Ready – Made my own image dataset with custom images. Whenever you are training a custom model the important thing is images. Yes, of course the images play a main role in deep learning. The accuracy of your model will be based on the training images. Dataset is the collection of specific data for your ML project needs. The type of data depends on the kind of AI you need to train. Basically, you have two datasets:

- Training
- Testing

Import Libraries – When we import modules, we're able to call functions that are not built into Python. Some modules are installed as part of Python, and some we will install through pip. Making use of modules allows us to make our programs more robust and powerful as we're leveraging existing code.

Load the Data – The source folder is the input parameter containing the images for different classes. Loading data in python environment is the most initial step of analysing data. Here, the pictures that I need to upload are being stored in the path.

Performed Data Pre-processing and Data Augmentation – To generate more images and batches of augmented data for Training_Data and Validation_Data. In order to make the most of our few training examples, we will "augment" them via a number of random transformations, so that our model would never see twice the exact same picture. This helps prevent overfitting and helps the model generalize better.

In Keras this can be done via the keras.preprocessing.image.ImageDataGenerator class. This class allows you to:

- configure random transformations and normalization operations to be done on your image data during training
- instantiate generators of augmented image batches (and their labels) via .flow(data, labels) or .flow_from_directory(directory). These generators can then be used with the Keras model methods that accept data generators as inputs, fit_generator, evaluate_generator and predict_generator.

These are just a few of the options available:

- rotation_range is a value in degrees (0-180), a range within which to randomly rotate pictures
- width_shift and height_shift are ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally
- rescale is a value by which we will multiply the data before any other processing. Our original images consist in RGB coefficients in the 0-255, but such values would be too high for our models to process (given a typical learning rate), so we target values between 0 and 1 instead by scaling with a 1/255 factor
- shear_range is for randomly applying shearing transformations
- zoom_range is for randomly zooming inside pictures
- horizontal_flip is for randomly flipping half of the images horizontally --relevant when there are no assumptions of horizontal asymmetry (e.g. real-world pictures)
- fill_mode is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift.

Now let's start generating some pictures using this tool and save them to a temporary directory, so we can get a feel for what our augmentation strategy is doing --we disable rescaling in this case to keep the images displayable. Let's prepare our data. We will use .flow_from_directory() to generate batches of image data (and their labels) directly from our jpgs in their respective folders. We can now use these generators to train our model. Therefore, transition from .flow_from_directory to .fit().

Data augmentation is one way to fight overfitting, but it isn't enough since our augmented samples are still highly correlated. Your main focus for fighting overfitting should be the entropic capacity of your model --how much information your model is allowed to store. A model that can store a lot of information has the potential to be more accurate by leveraging more features, but it is also more at risk to start storing irrelevant features. Meanwhile, a model that can only store a few features will have to focus on the most significant features found in the data, and these are more likely to be truly relevant and to generalize better.

Visualize images generated to gain insights – Plotting leads to better decision making and clear picture of data.

Created a model with .h5 extension – Saving our model in .h5 file as this means that we can load and use the model directly, without having to re-compile it.

Created Checkpoints – The checkpoint may be used directly, or used as the starting point for a new run, picking up where it left off. When training deep learning models, the checkpoint is the weights of the model. These weights can be used to make predictions as is, or used as the basis for ongoing training.

Built the CNN Model – Here used, CNN for Image Classification. CNN image classifications take an input image, process it and classify it under certain categories (Eg., Dog, Cat, Tiger, Lion). Computers sees an input image as array of pixels and it depends on the image resolution. Based on the image resolution, it will see h x w x d (h = Height, w = Width, d = Dimension).

In Keras, you create 2D convolutional layers using the keras.layers.Conv2D() function.

A convolution layer “scans” A source image with a filter of, for example, 5×5 pixels, to extract features which may be important for classification. This filter is also called the convolution kernel. The kernel also contains weights, which are tuned in the training of the model to achieve the most accurate predictions.

In a 5×5 kernel, for each 5×5pixel region, the model computes the dot products between the image pixel values and the weights defined in the filter.

A 2D convolution layer means that the input of the convolution operation is three-dimensional, for example, a colour image which has a value for each pixel across three layers: red, blue and green. However, it is called a “2D convolution” because the movement of the filter across the image happens in two dimensions. The filter is run across the image three times, once for each of the three layers.

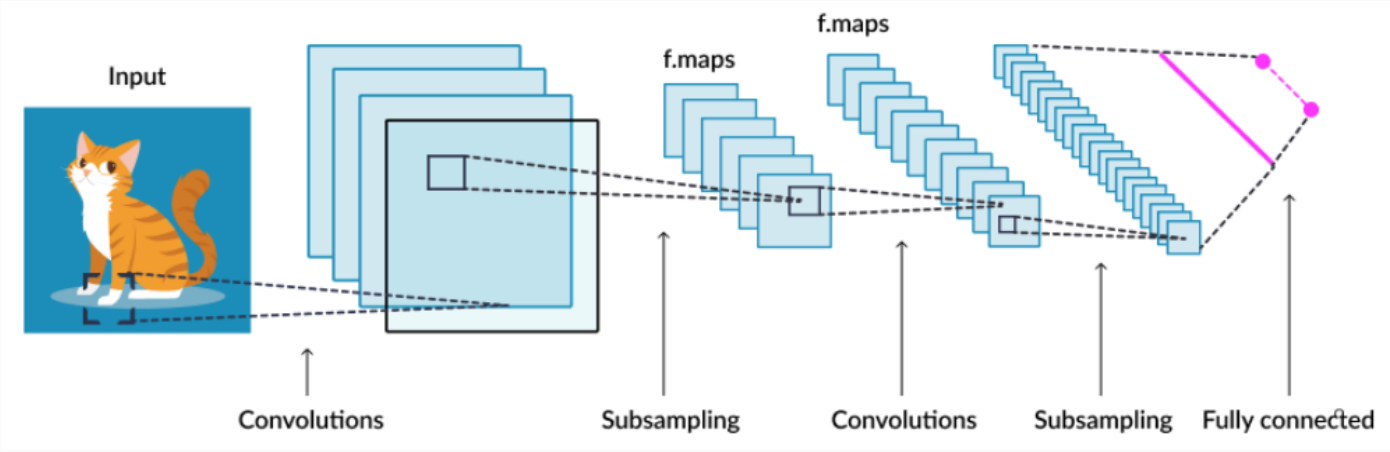
After the convolution ends, the features are down-sampled, and then the same convolutional structure repeats again. At first, the convolution identifies features in the original image (for example in a cat, the body, legs, tail, head), then it identifies sub-features within smaller parts of the image (for example, within the head, the ears, whiskers, eyes). Eventually, this process is meant to identify the essential features that can help classify the image.

Here is the full signature of the Keras Conv2D function:

```
kernel_size, strides=(1, 1), padding='valid', data_format=None,
dilation_rate=(1, 1), activation=None, use_bias=True,
kernel_initializer='glorot_uniform', bias_initializer='zeros',
```

```
kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,
kernel_constraint=None, bias_constraint=None)
```

CNN Architecture:



I find that the pretrained models work best.

And you can find all of the keras applications over here: https://keras.io/api/applications/

I create my own CNN architecture and it works well on the training and as well as testing dataset.

Overfitting happens when a model exposed to too few examples learns patterns that do not generalize to new data, i.e. when the model starts using irrelevant features for making predictions. For instance, if you, as a human, only see three images of people who are lumberjacks, and three, images of people who are sailors, and among them only one lumberjack wears a cap, you might start thinking that wearing a cap is a sign of being a lumberjack as opposed to a sailor. You would then make a pretty lousy lumberjack/sailor classifier.

There are different ways to modulate entropic capacity. The main one is the choice of the number of parameters in your model, that is, the number of layers and the size of each layer. Another way is the use of weight regularization, such as L1 or L2 regularization, which consists in forcing model weights to taker smaller values.

In our case we will use a very small convnet with few layers and few filters per layer, alongside data augmentation and dropout. Dropout also helps reduce overfitting, by preventing a layer from seeing twice the exact same pattern, thus acting in a way analogous to data augmentation (you could say that both dropout and data augmentation tend to disrupt random correlations occurring in your data).

When updating the curve, to know in which direction and how much to change or update the curve depending upon the slope. That is why we use differentiation in almost every part of Machine Learning and Deep Learning.

ReLU (Rectified Linear Unit) Activation Function, as you can see, the ReLU is half rectified (from bottom). f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero.

Range: [0 to infinity]

The function and its derivative both are monotonic.

But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.

The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time.

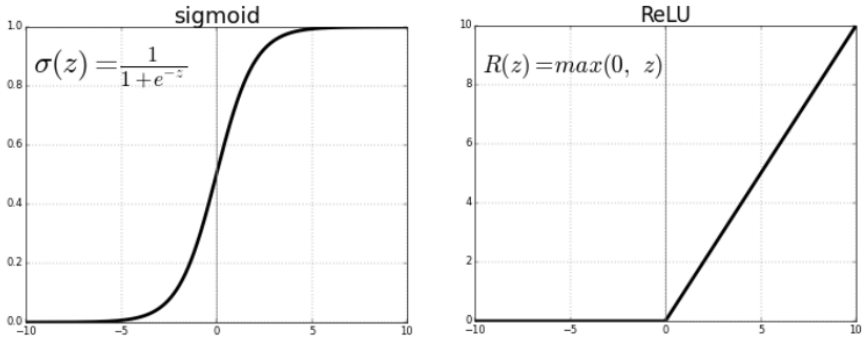
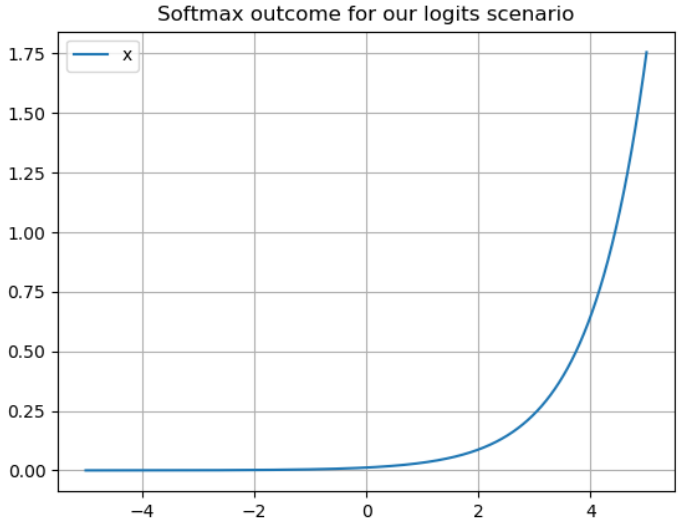


Fig: ReLU v/s Logistic Sigmoid

Derivative or Differential: Change in y-axis w.r.t. change in x-axis.It is also known as slope.
Monotonic function: A function which is either entirely non-increasing or non-decreasing.
Softmax is used as the activation function for multi-class classification problems where class membership is required on more than two class labels. Softmax is differentiable and it allows us to optimize a cost function. Softmax is exponential and enlarges differences - push one result closer to 1 while another closer to 0. It turns scores aka logits into probabilities. Cross entropy (cost function) is often computed for output of softmax and true labels (encoded in one hot encoding).



Compiled the CNN Model – Now we will compile our model with the loss, optimizer and the metrics of evaluation.

Trained the CNN Model on Training and Validation Data – I would recommend to use GPU as I have used 500 epochs to get 98% accuracy.

Checked Accuracy through Visualization – Accuracy is the number of correctly predicted data points out of all the data points. More formally, it is defined as the number of true positives and true negatives divided by the number of true positives, true negatives, false positives, and false negatives. Model Visualization provides reason and logic behind to enable the accountability and transparency on the model. The only accuracy will not give the exact interpretation of the model. Test model accuracy and trust that the classifier or model is working correctly. It also helps to debug models.

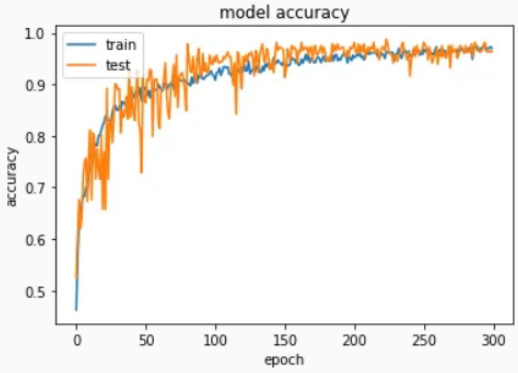
- Proper explanations of what the model is doing?
- Why the results are what they are?
- Output in a form or visual form described to the non-technical persons.

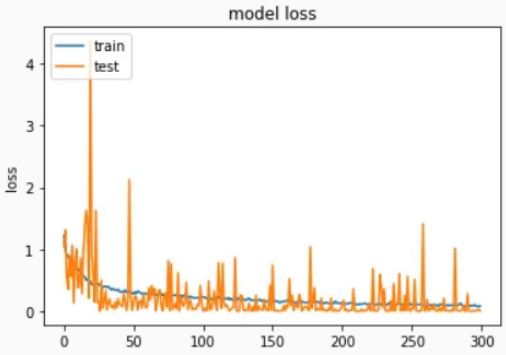
Created Web App – Created web app in flask for end-users.

Checking the Model Visualization

- Basic Model Evaluation Graphs

Model Accuracy & Loss





Creation of App

Here, I am creating Flask App. Loading the model that we saved then converting an image to array and normalizing it.
Getting index of max value.
Get input image from client then predict class and render respective .html page for solution.
You can create it with Streamlit also.

Technical Aspect

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine. Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow.
Matplotlib is used for EDA. Visualization of graphs helps to understand data in better way than numbers in table format. Matplotlib is mainly deployed for basic plotting. It consists of bars, pies, lines, scatter plots and so on. Inline command display visualization inline within frontends like in Jupyter Notebook, directly below the code cell that produced it.
Flask is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.
Numpy used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. It contains a multi-dimensional array and matrix data structures. It can be utilised to perform a number of mathematical operations on arrays.

Installation

Using intel core i5 9th generation with NVIDIA GFORECE GTX1650.
Windows 10 Environment Used.
Already Installed Anaconda Navigator for Python 3.x
The Code is written in Python 3.8.
If you don't have Python installed then please install Anaconda Navigator from its official site.
If you are using a lower version of Python you can upgrade using the pip package, ensuring you have the latest version of pip, *python* -m pip install --*upgrade pip and press Enter*.

Run/How to Use/Steps

Keep your internet connection on while running or accessing files and throughout too.
Follow this when you want to perform from scratch.

Open Anaconda Prompt, Perform the following steps:

cd <PATH>
pip install tensorflow==2.1.0
pip install Keras==2.4.3
pip install numpy==1.18.5
pip install flask==1.1.2
Note: If it shows error as 'No Module Found' , then install relevant module.

You can also create requirement.txt file as, pip freeze > requirements.txt

Create Virtual Environment:
conda create -n cotton python=3.6
y
conda activate cotton
cd <PATH-TO-FOLDER>

run .py or .ipynb files.
Paste URL to browser to check whether working locally or not.

Follow this when you want to just perform on local machine.
Download ZIP File.
Right-Click on ZIP file in download section and select Extract file option, which will unzip file.
Move unzip folder to desired folder/location be it D drive or desktop etc.
Open Anaconda Prompt, write cd <PATH> and press Enter.
eg: cd C:\Users\Monica\Desktop\Projects\Python Projects 1\23)End_To_End_Projects\Project_7_DL_FileUse_EndToEnd_CottonDiseasePrediction\Project_DL_CottonDiseasePrediction

conda create -n cotton python=3.6
y
conda activate cotton
In Anconda Prompt, pip install -r requirements.txt to install all packages.
In Anconda Prompt, pip install keras
In Anconda Prompt, write python app.py and press Enter.
Paste URL 'http://127.0.0.1:5000/' to browser to check whether working locally or not.

Please be careful with spellings or numbers while typing filename and easier is just copy filename and then run it to avoid any silly errors.
Note: cd <PATH>
[Go to Folder where file is. Select the path from top and right-click and select copy option and paste it next to cd one space <path> and press enter, then you can access all files of that folder]
[cd means change directory]

Directory Tree/Structure of Project

```
Project_DL_CottonDiseasePrediction/
├── Images_screenshot/
├── Datasets/
│   ├── test/
│   ├── train/
│   └── val/
├── model/
│   └── v3_pred_cott_dis.h5
├── static/
│   ├── images/
│   └── user uploaded/
├── templates/
│   ├── disease_plant.html
│   ├── healthy_plant.html
│   ├── healthy_plant_leaf.html
│   └── index.html
├── Sample_Images_To_Test_App/
├── cotton_disease_prediction_final.py
├── Cotton_Disease_Prediction_Final.ipynb
├── app.py
└── requirements.txt
```

To Do/Future Scope

Can Deploy on AWS.

Technologies Used/System Requirements/Tech Stack

Download the Material

You can Download Dataset from here: https://github.com/monicadesAI-tech/Project_74/tree/main/Datasets

You can Download Entire Project from here: https://github.com/monicadesAI-tech/Project_74

You can Download Model Building from here: https://github.com/monicadesAI-tech/Project_74/blob/main/cotton_disease_prediction_final.py

You can download Google Colab File from here: https://github.com/monicadesAI-tech/Project_74/blob/main/Cotton_Disease_Prediction_Final.ipynb

You can Download App_File from here: https://github.com/monicadesAI-tech/Project_74/blob/main/app.py

You can see Detailed Project at Website here: https://github.com/monicadesAI-tech.github.io/cottonfinal.html

Download saved model from here: https://github.com/monicadesAI-tech/Project_74/tree/main/model

Download dependencies from here: https://github.com/monicadesAI-tech/Project_74/blob/main/requirements.txt

Download images for test from here: https://github.com/monicadesAI-tech/Project_74/tree/main/Sample_Images_To_Test_App

Conclusion

- Modelling

Using pre-trained network, **s**uch a network would have already learned features that are useful. The reason why we are storing the features offline rather than adding our fully-connected model directly on top of a frozen convolutional base and running the whole thing, is computational efficiency, so we can do in 20 epochs rather than 500 epochs.

- Analysis

It gives me more than 98% accuracy on training and validation data set in just 500 epochs. I am trying to increase accuracy with more data and epochs.

Credits

Krish Naik
IndianAIProduction

Paper Citation

https://www.researchgate.net/publication/290655824_Identification_of_cotton_diseases_based_on_cross_information_gain_deep_forward_neural_network_classifier_with_PSO_feature_selection