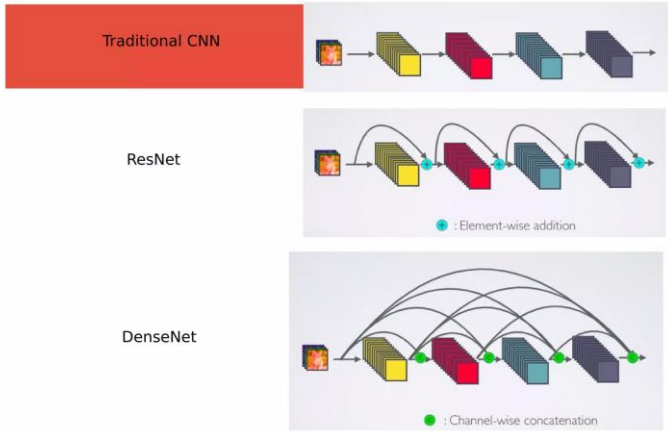
CNNs and Advanced DL methods

Dr. Huseyin Kusetogullari

DenseNet

- DenseNets require fewer parameters than an equivalent traditional CNN, as there is no need to learn redundant feature maps.
- Traditional feed-forward neural networks connect the output of the layer to the next layer after applying a ***composite of operations***.
- **DenseNets do not sum the output feature maps of the layer with the incoming feature maps but concatenate them.**

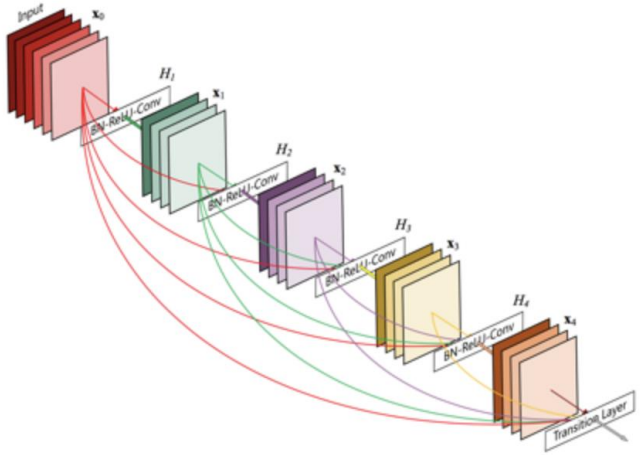


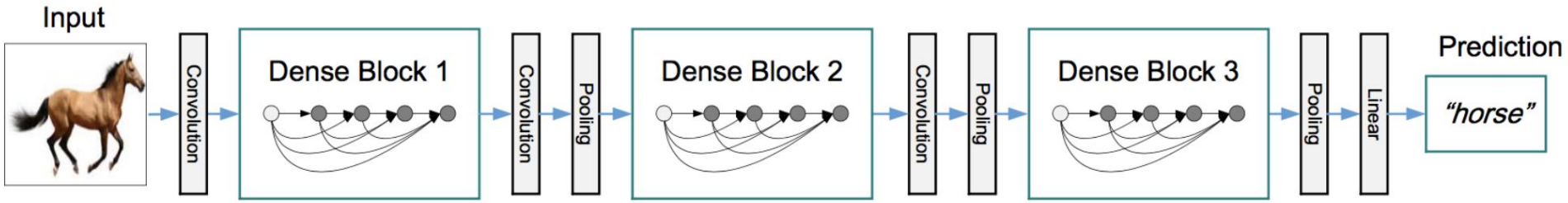
DenseNet

- DenseNets are divided into **DenseBlocks, where the dimensions of the feature maps remains constant within a block, but the number of filters changes between them**.
- These layers between them are called ***Transition Layers*** and take care of the downsampling applying a batch normalization, a 1x1 convolution and a 2x2 pooling layers.
- A **transition layer** is used to reduce the number of feature maps before passing them on to the next block of densely connected layers.

DenseNet

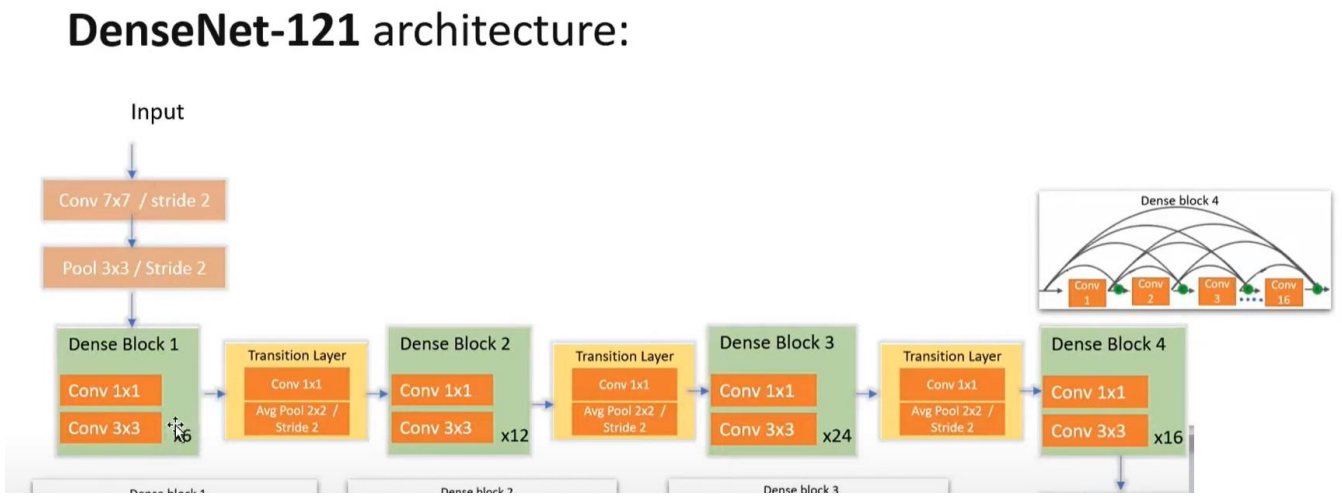
Concatenate output of a layer to all successive layers





DenseNet-121 Architecture

- In a DenseNet architecture, a transition layer has a specific role between the dense blocks.
- Dense blocks in DenseNet consist of layers where each layer is connected to every other layer in a feed-forward fashion.
- However, as the network gets deeper, the size of the feature-maps can become quite large.
- To manage this and to improve computational efficiency, transition layers are used.



DenseNet-121 Architecture

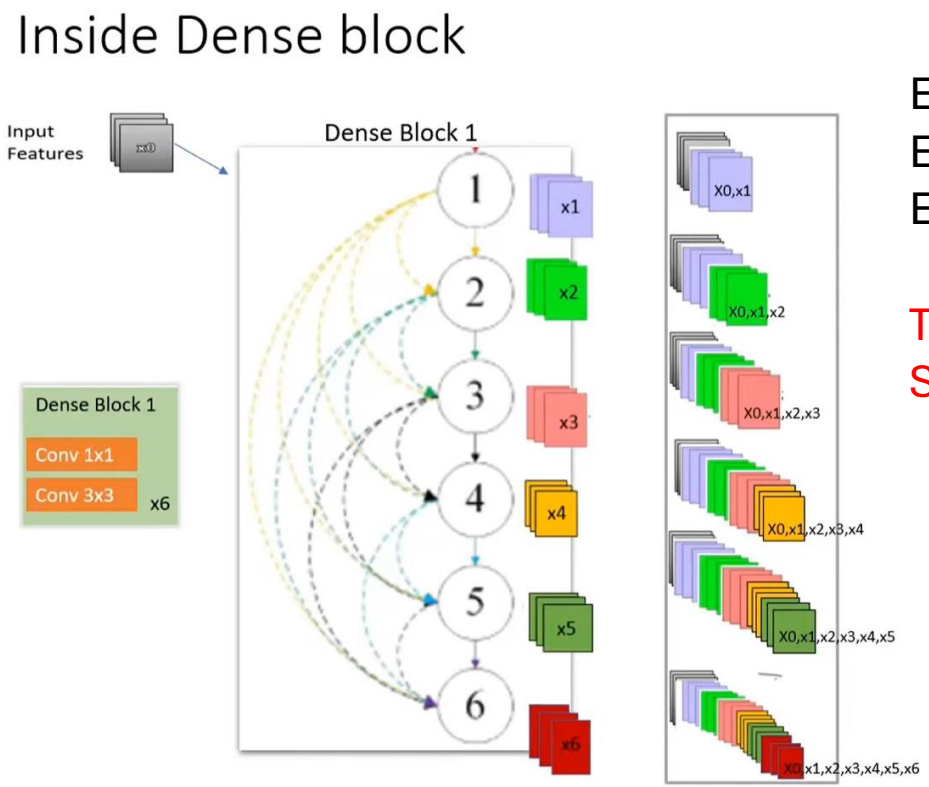
In **DenseNet-121**, the number of convolutional layers in each DenseBlock is:

- **DenseBlock1**: 6 layers
- **DenseBlock2**: 12 layers
- **DenseBlock3**: 24 layers
- **DenseBlock4**: 16 layers

These numbers can vary depending on whether you're using **DenseNet-121**, **DenseNet-169**, **DenseNet-201**, etc.

Component	Count
Initial Conv layer	1
DenseBlock 1: 6 × 2 convs	12
Transition 1: 1 conv	1
DenseBlock 2: 12 × 2 convs	24
Transition 2: 1 conv	1
DenseBlock 3: 24 × 2 convs	48
Transition 3: 1 conv	1
DenseBlock 4: 16 × 2 convs	32
Classification FC layer	1
Total	**121**

Inside Dense Block



Each layer is connected to every other layer
Each layer receives the feature-maps
Each layer adds some features on top of the existing feature maps

To perform the concatenation operation, we need to make sure that th
Size of the feature maps that we are concanating is the same.

DenseNet

Advantages:

1.Feature Reuse: DenseNet connects each layer to every other layer in a feed-forward fashion, promoting feature reuse and alleviating the vanishing-gradient problem.

2.Parameter Efficiency: It requires fewer parameters compared to traditional architectures like VGG and ResNet, thanks to feature concatenation instead of addition.

3.Strong Feature Propagation: Dense connectivity facilitates strong feature propagation through the network, enabling effective learning of feature representations.

Disadvantages:

1.Memory Consumption: Dense connectivity leads to increased memory consumption during training and inference, which can be a limiting factor for deployment on resource-constrained devices.

2.Computationally Intensive: The dense connectivity pattern results in a higher computational cost compared to traditional architectures, especially as the network depth increases.

3.Difficulty in Interpretability: The dense connections can make it challenging to interpret the learned representations compared to more traditional architectures.

Recurrent Neural Networks

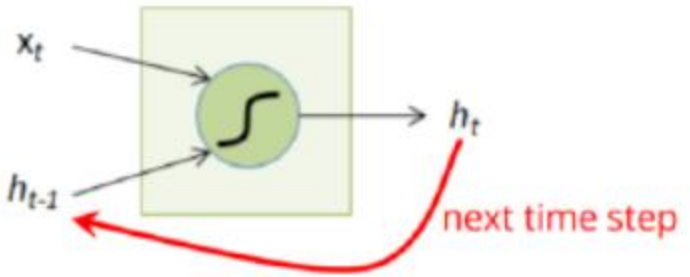
- A recurrent neural network (RNN) is a class of artificial neural network.
- A standard feed-forward neural network computes its output values in a sequential input-process-output manner.
- Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs.
- RNNs are very powerful, because they combine two properties:
  - Distributed hidden state that allows them to store a lot of information about the past efficiently.
  - Non-linear dynamics that allows them to update their hidden state in complicated ways.

- With enough neurons and time, RNNs can compute anything that can be computed by your computer.

Cont.

The recurrent neuron is shown below:

- x_t: Input at time t
- h_{t-1}: State at time t-1



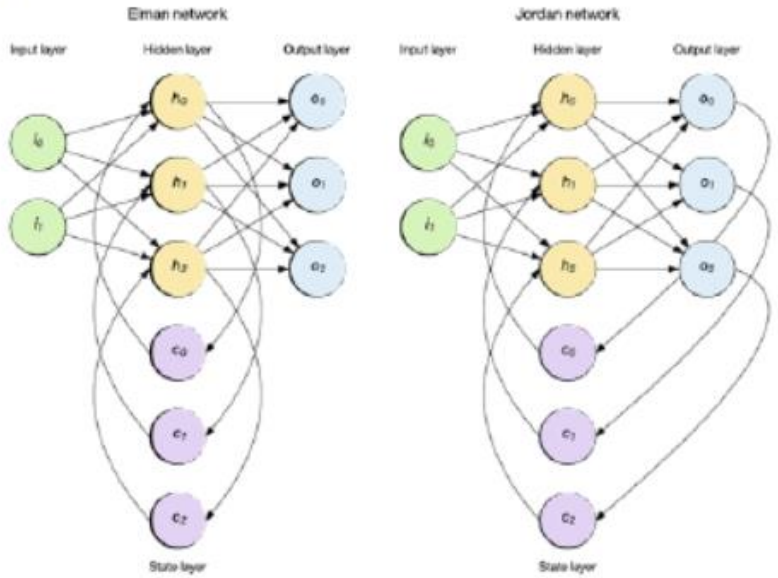
$$h_t = f(W_h h_{t-1} + W_x x_t)$$

Types of RNNs

- Elman Network
- Jordan Network
- Hopfield
- Bidirectional associative memory
- Independent RNN (IndRNN)
- Second order RNNs
- Long short-term memory
- Recurrent multilayer perceptron network
- And many others

Cont.

- Applications of **time-series data** require a new type of topology that can consider the **history** of the input. This is where RNNs can be applicable and very useful.
- Two popular recurrent network approaches are shown below:



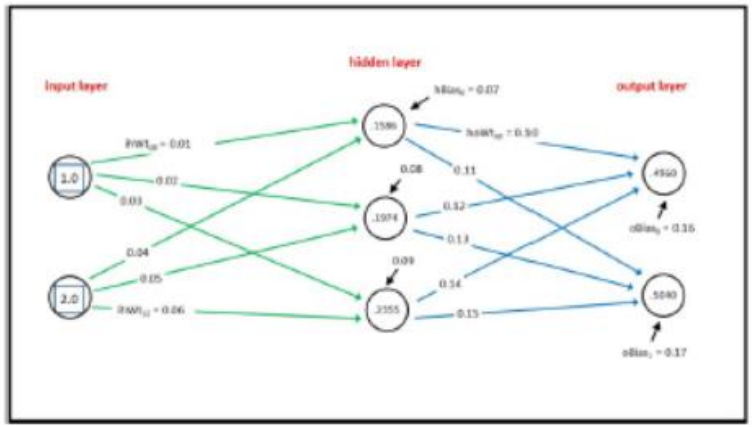
In the **Elman RNN**, the hidden layer feeds a state layer of context nodes that retain memory of past inputs. As shown in the following figure, a single set of context nodes exists that maintains memory of the prior hidden layer result.

In **Jordan RNN**, it stores the output layer into the state layer instead of maintaining history of the hidden layer.

- The state layer influences the next stage of input and therefore can be applied to time-varying patterns of data.

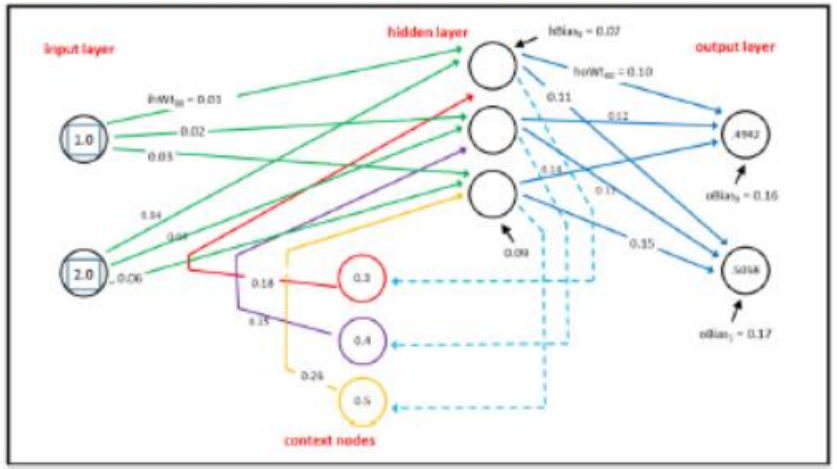
Elman RNN

- **Figure** shows a standard feed-forward neural network with two input nodes, three hidden nodes and two output nodes.
- The network has a total of 17 weights and biases.
- There are 2 * 3 = 6 input-to-hidden weights with values 0.01 through 0.06.
- There are 3 hidden node biases with values 0.07, 0.08 and 0.09.
- There are 3 * 2 = 6 hidden-to-output weights with values 0.10 through 0.15.
- And there are 2 output node biases with values 0.16 and 0.17.
- The neural network has input values of 1.0 and 2.0.



Cont.

- An obvious difference between a regular neural network and a recurrent neural network is that a recurrent network has more connections due to an additional set of nodes that are usually called **context nodes** or **context units**.
- There is a one-to-one correspondence between the hidden nodes and the context nodes. Because the network in the diagram has three hidden nodes, there are three context nodes.
- There is a feedback loop from each hidden node to its corresponding context node. In **Figure** these feedback loops are indicated by the blue dashed arrows.
- Each context node is forward-connected to corresponding hidden node with an associated weight. These weights are indicated by the red, purple and orange arrows.

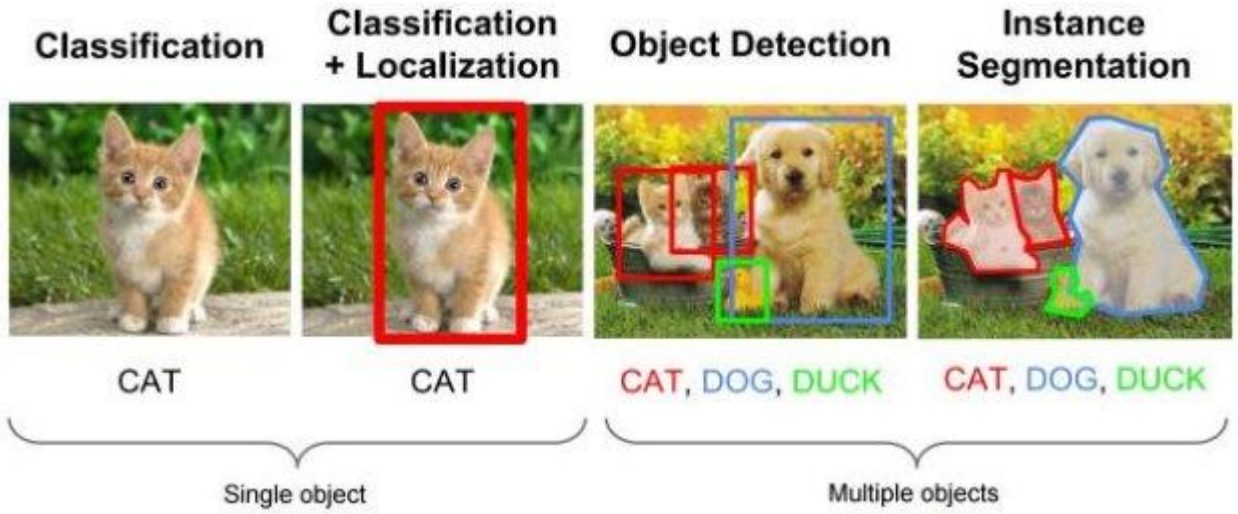


Cont.

- n high-level pseudo code, the output values for a recurrent neural network are calculated like so:
  - *Compute preliminary hidden node sums using input values and context node values*
  - *Apply activation function to hidden node sums to get hidden node values*
  - *Compute preliminary output node sums using hidden node values*
  - *Apply activation function to output node sums to get final output node values*
  - *Copy hidden node values to associated context nodes for use with next input*

What is YOLO?

- **YOLO** stands for **You Only Look Once**, and it's a real-time object detection algorithm. That means it can both **identify what an object is** (classification) and **where it is** (localization) in an image



- Instead of looking at parts of the image multiple times (like traditional methods), YOLO looks at the entire image once and predicts everything at once.
- It breaks the image into a grid and each grid cell predicts:
- Whether there's an object.
- The object's class (e.g., dog, car).
- The coordinates of a bounding box around it.

YOLOv1 (2016)
•**Total layers**: ~24 convolutional + 2 fully connected layers
•**Architecture**: Inspired by GoogLeNet (Inception-style), ends with 2 FC layers

YOLOv2 (YOLO9000, 2017)
•**Total layers**: ~30 convolutional layers + 1 passthrough layer
•**Backbone**: Darknet-19 (19 conv layers + 5 maxpool layers)
•**Changes**: Removed fully connected layers; used anchor boxes

YOLOv3 (2018)
•**Total layers**: ~106 layers
•**Backbone**: Darknet-53 (53 conv layers, deeper than v2)
•**Features**: Multi-scale prediction (3 scales)

YOLOv4 (2020)
•**Total layers**: ~162 layers (varies based on configuration)
•**Backbone**: CSPDarknet53
•**Neck**: SPP + PAN
•**Head**: YOLO detection layers

•YOLOv5 , YOLOv6 (by Meituan), YOLOv7 (2022)

YOLOv8 (2023, Ultralytics)
•**Total layers**: Varies by model (n, s, m, l, x)
•**Architecture**: Custom Ultralytics backbone (not Darknet-based anymore)
•**Framework**: PyTorch
•**Variants**:

- YOLOv8n (Nano): ~168 layers
- YOLOv8s (Small): ~225 layers
- YOLOv8m (Medium): ~308 layers
- YOLOv8l (Large): ~393 layers
- YOLOv8x (X-Large): ~489 layers

How YOLO Works - Step by Step

Let's say we input an image of size **416x416 pixels**:

1. **Divide the image into a grid**, e.g., 13x13 cells.
2. Each grid cell is responsible for detecting objects **whose center falls inside it**.
3. Each cell outputs:
    - Bounding box coordinates: x, y, w, h
    - Confidence score (how sure it is there's an object)
    - Class probabilities (e.g., 90% dog, 5% cat)
4. All these predictions are combined, and non-max suppression is used to **remove overlapping boxes** and keep only the best ones.

YOLO was originally introduced by Joseph Redmon in 2016. Since then, we've had YOLOv2, YOLOv3, YOLOv4, YOLOv5 (by Ultralytics), YOLOv6, YOLOv7, YOLOv8... and each version gets faster and smarter!

It is lightweight, modular, and optimized for deployment.

Key Features

- Fast inference speed
- High accuracy
- Scalable (YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x – from small to extra large)
- Built-in support for training, validation, testing

Model	Parameters	Speed (ms)	mAP@0.5
YOLOv5n	1.9M	Fastest	Lower
YOLOv5s	7.2M	Fast	Good
YOLOv5m	21.2M	Medium	Better
YOLOv5l	46.5M	Slower	High
YOLOv5x	87.7M	Slowest	Highest

The architecture of YOLOv5 can be divided into **three main parts**:

1. Backbone

Extracts important features from the input image.

- **Focus layer**: Slices input image and concatenates slices along channel dimension, reducing spatial resolution and increasing depth.
- **CSPDarknet**: A modified version of **Darknet-53** using Cross Stage Partial (CSP) connections to reduce computation and improve learning capability.
- Contains:
    - Convolution layers
    - Bottleneck layers with residual connections
    - CSP blocks

2. Neck

Combines and refines features from the backbone.

- Uses a combination of:
    - **PANet (Path Aggregation Network)**: Enhances feature fusion across different scales.
    - **FPN (Feature Pyramid Network)**: Helps in detecting objects at multiple scales.

3. Head

Performs final object classification and bounding box regression.

- Generates predictions at 3 different scales (small, medium, large objects)
- Output includes:
    - **Bounding box coordinates**
    - **Objectness score**
    - **Class probabilities**

Annotating Images for YOLO

0 0.612264 0.463443 0.511792 0.879717

0 — Class ID (this refers to the object category; 0 is usually the first class in your list of labels)

0.612264 — x-center of the bounding box (relative to image width, so it’s normalized between 0 and 1)

0.463443 — y-center of the bounding box (relative to image height)

0.511792 — width of the bounding box (again, relative to image width)

0.879717 — height of the bounding box (relative to image height)

How to collect a custom dataset and annotate it?

How to train a YOLO algorithm on the created custom dataset?

Start-Up Idea

PROJECT: Automated Chicken Detection, Localization, and Behavior Analysis in Farm Environments

Discussion

Lab

- Building a various classifiers for Image Recognition

Project Discussion and Questions???

Questions?