



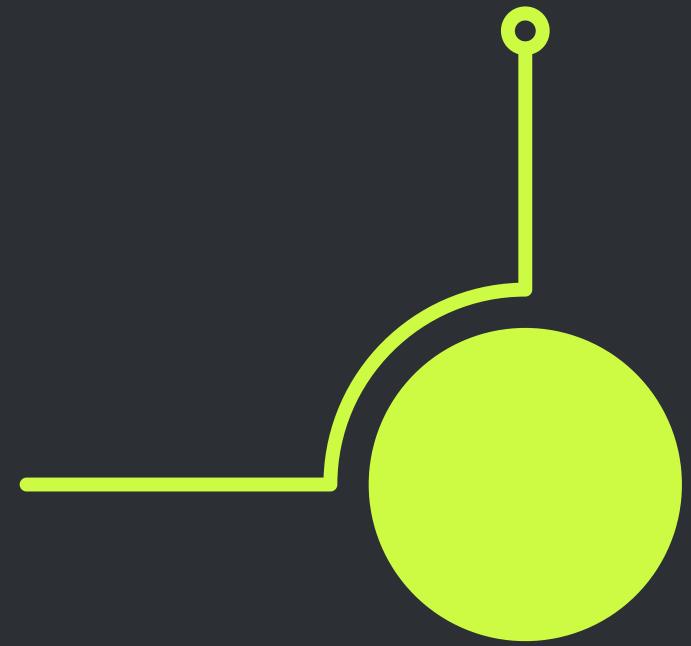
# Lunch time

Machine Vision  
using Python (MVUP01)



# Morning Session (9:30 - 12:00)

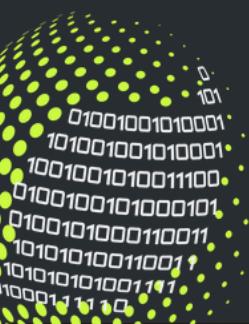
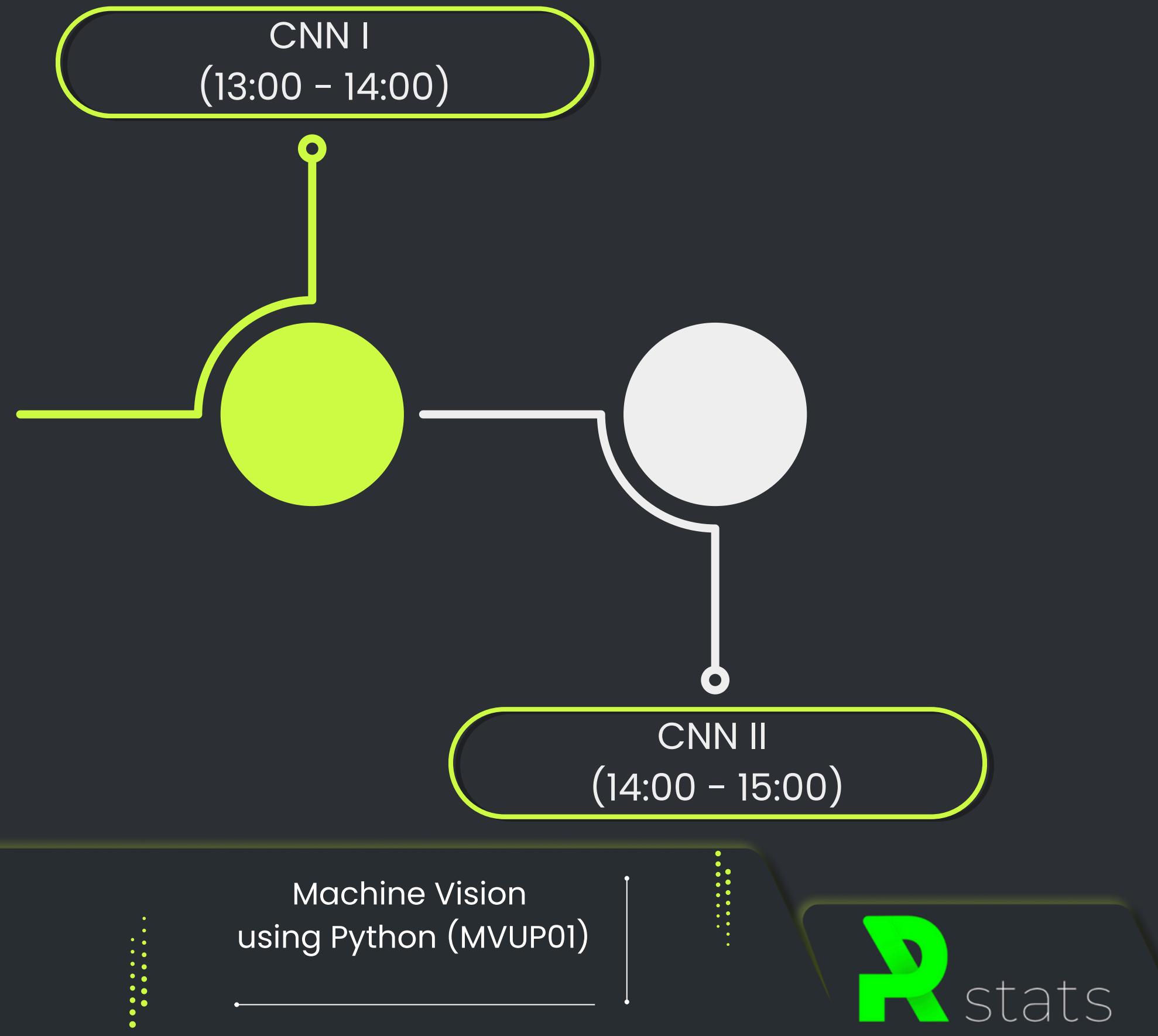
CNN I  
(13:00 - 14:00)



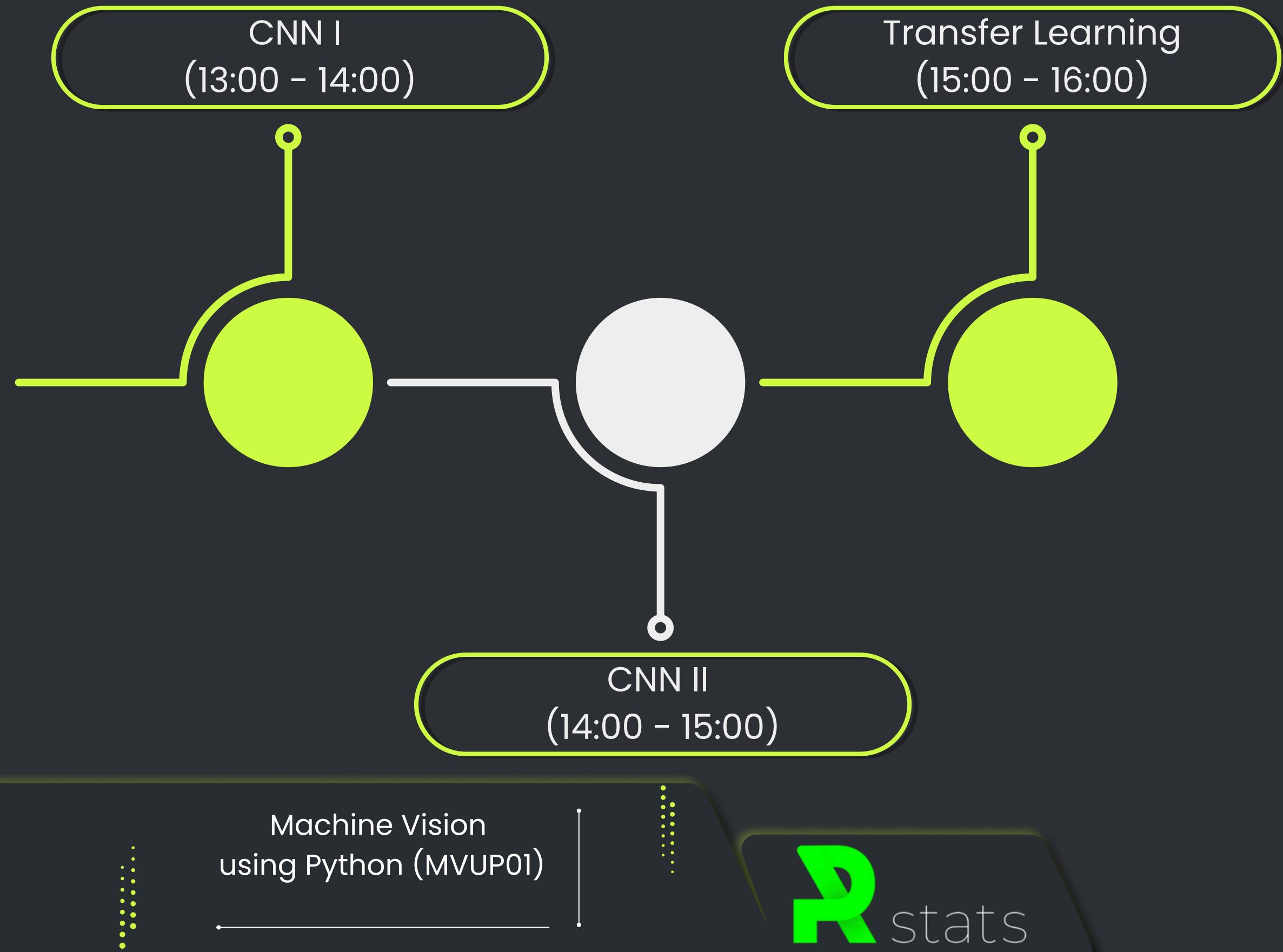
Machine Vision  
using Python (MVUP01)



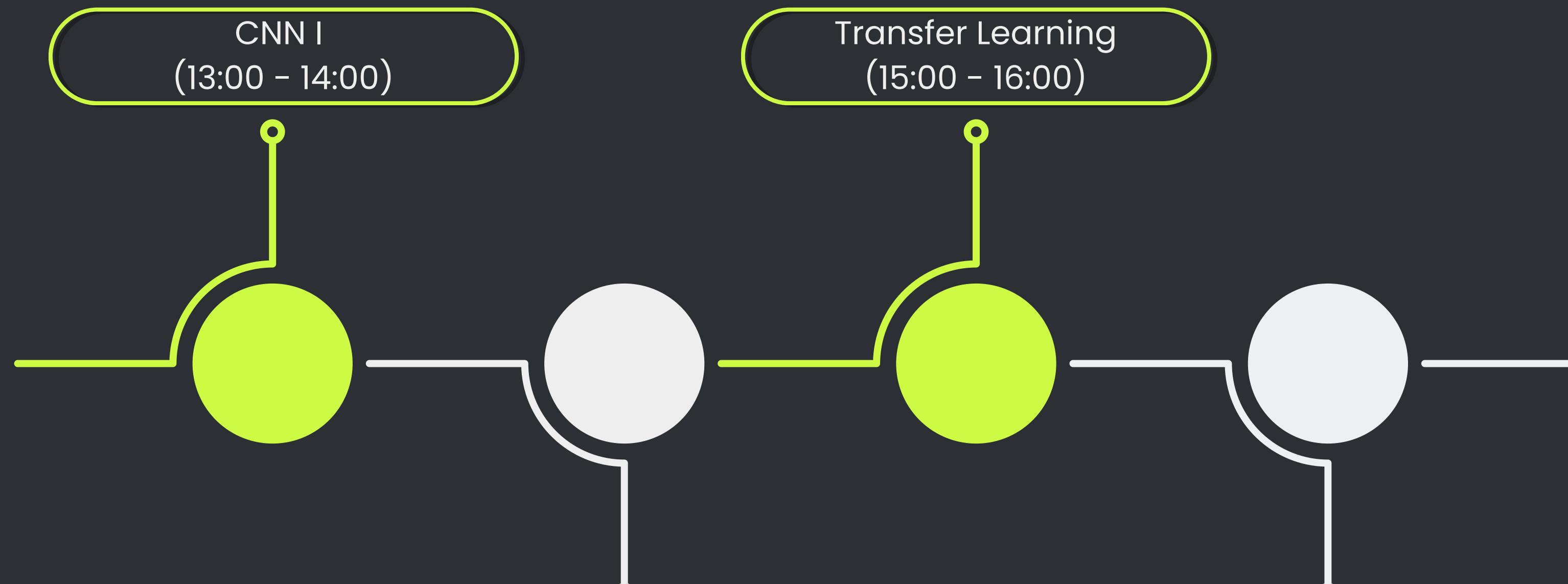
# Morning Session (9:30 - 12:00)



# Morning Session (9:30 - 12:00)



# Afternoon Session (13:00 - 17:30)



Machine Vision  
using Python (MVUP01)

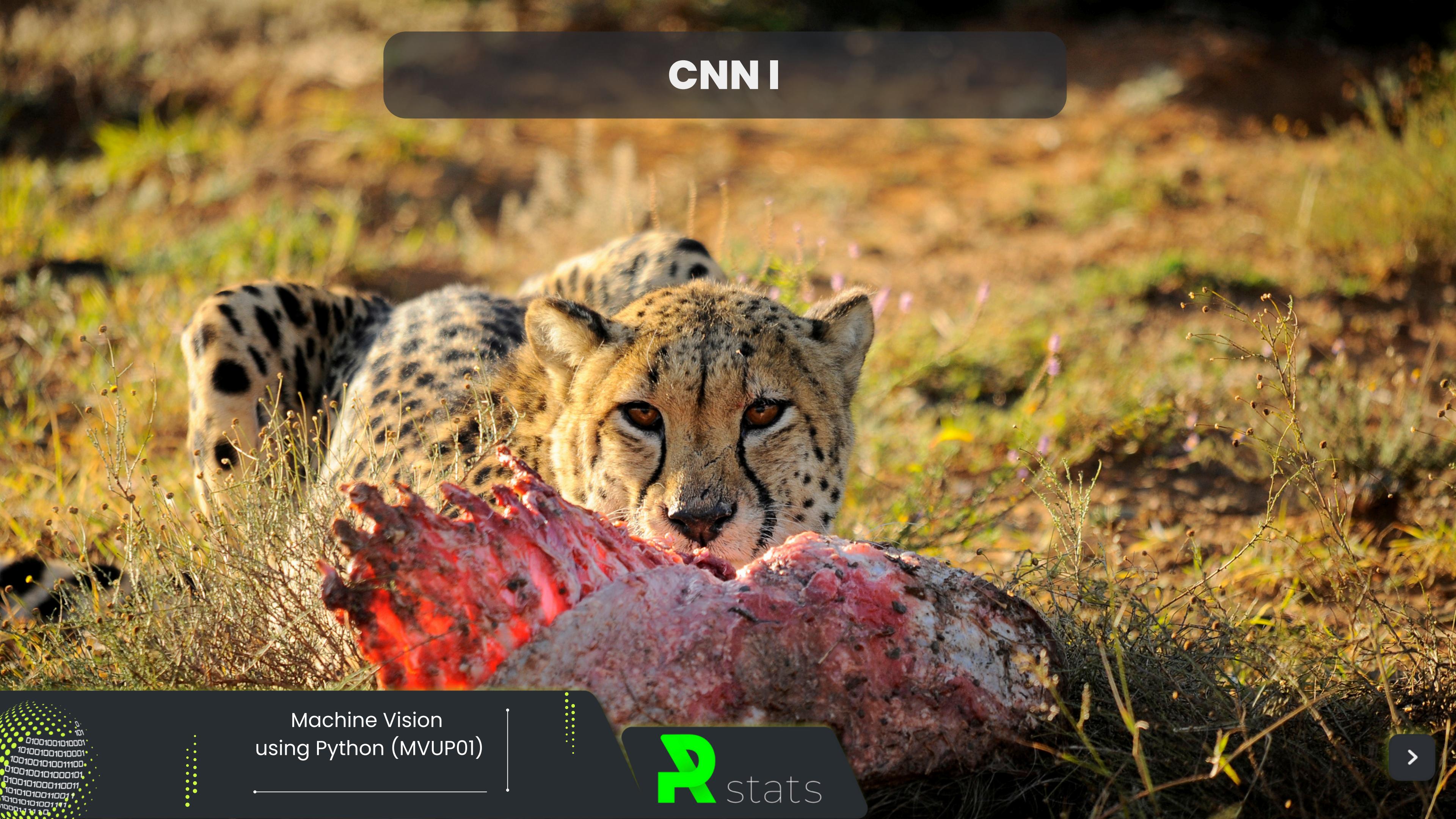




CNN I

Machine Vision  
using Python (MVUP01)



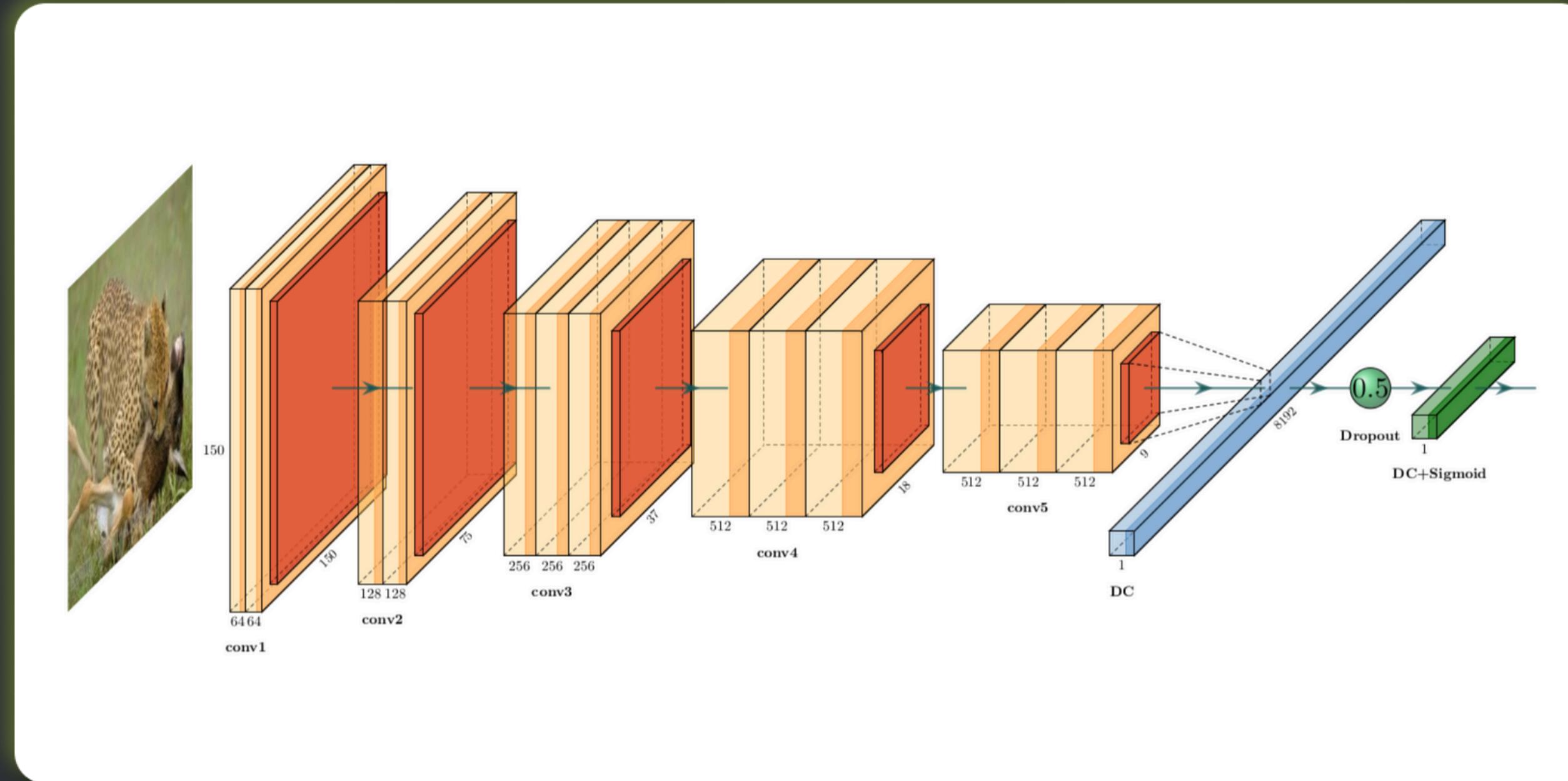
A close-up photograph of a cheetah's head and upper body, positioned behind a large, raw piece of red meat. The cheetah is looking directly at the camera with a focused expression. The background is a blurred savanna landscape with green grass and small purple flowers. A dark, semi-transparent rectangular overlay is centered at the top of the image, containing the text "CNN I".

CNN I

Machine Vision  
using Python (MVUP01)



# CNN I



Machine Vision  
using Python (MVUP01)



# CNN I



Machine Vision  
using Python (MVUP01)

R stats



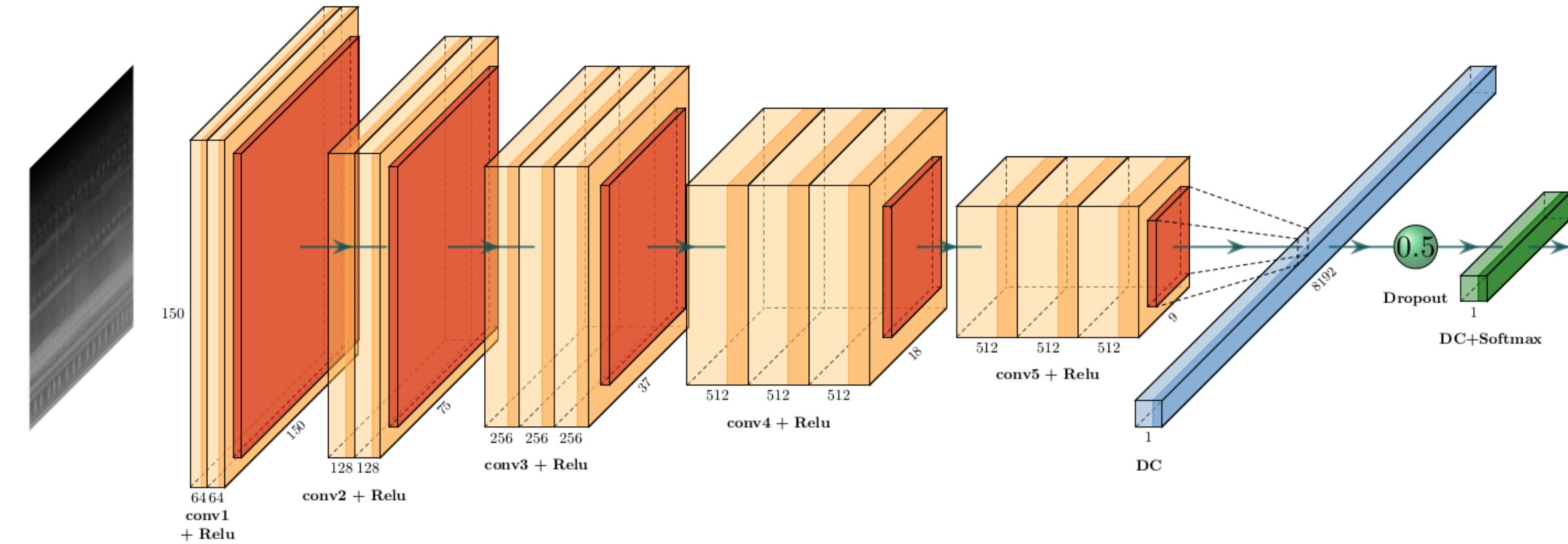
A close-up photograph of two barn owls perched on a dark, textured tree branch. The owls have light-colored feathers with dark brown spots and distinct 'ear tufts'. They are facing each other, with their heads touching. The background consists of blurred green and yellow leaves.

CNN I

Machine Vision  
using Python (MVUP01)



# CNN I



Machine Vision  
using Python (MVUP01)



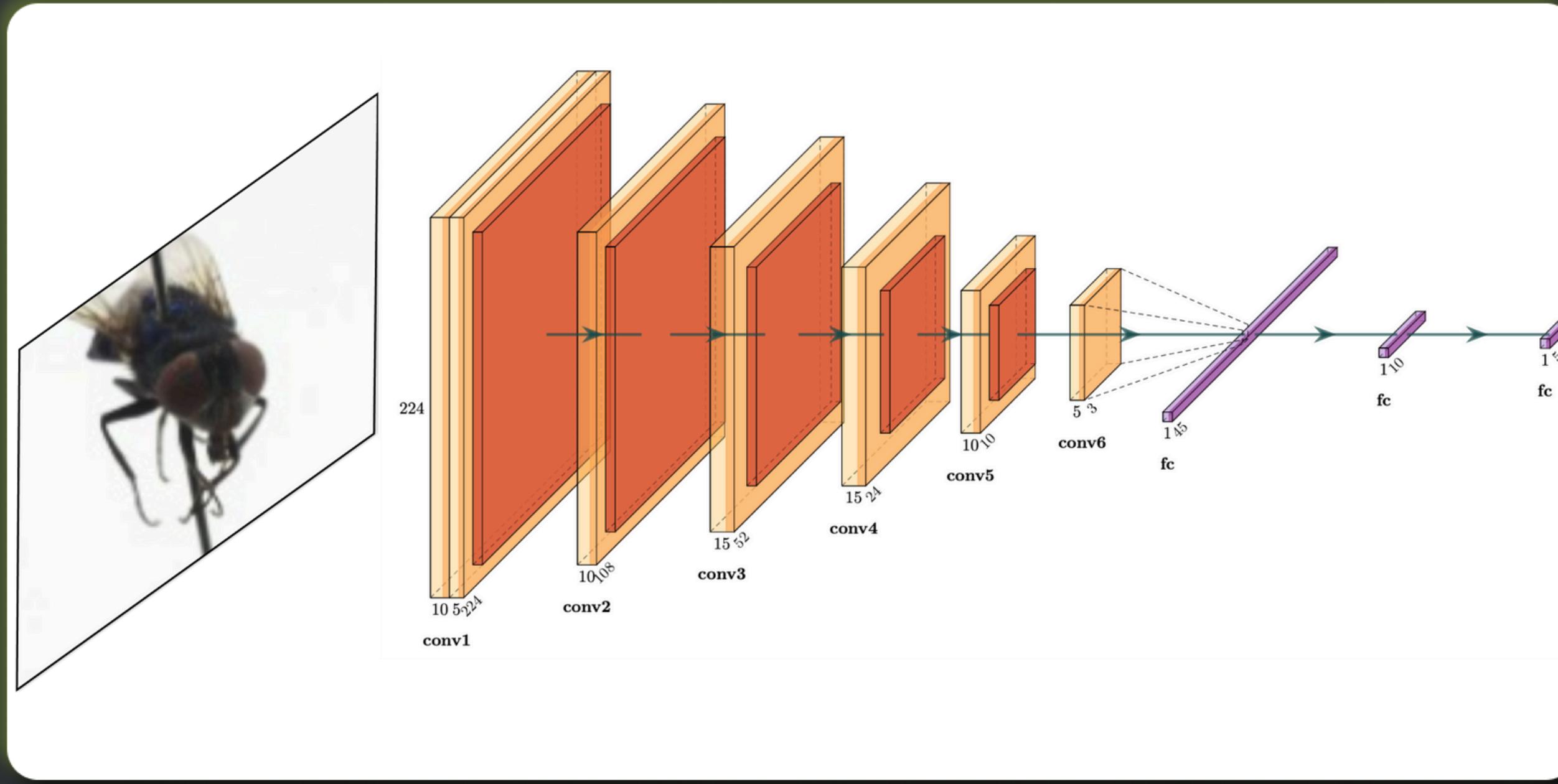


# CNN I

Machine Vision  
using Python (MVUP01)



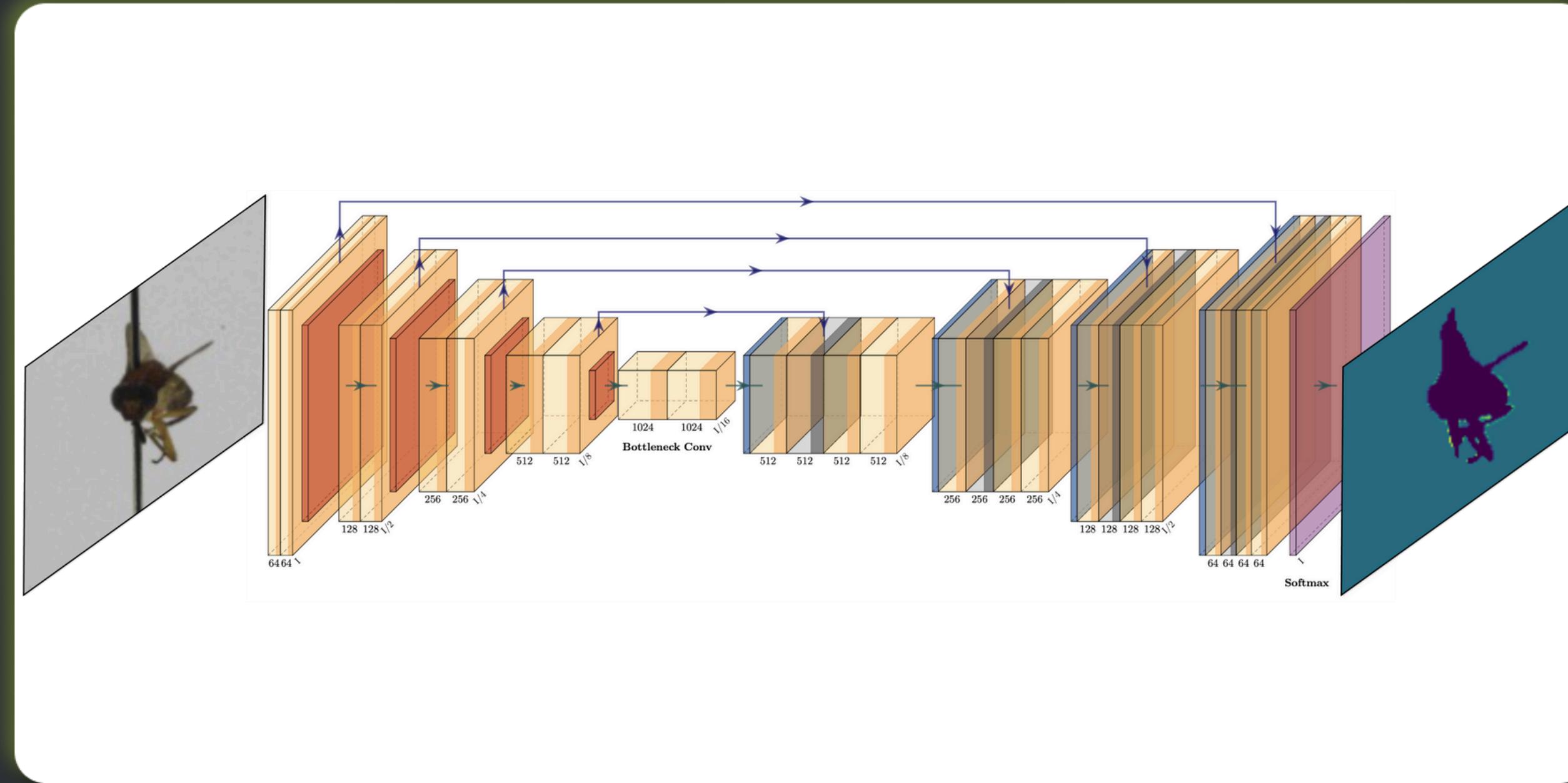
# CNN I



Machine Vision  
using Python (MVUP01)



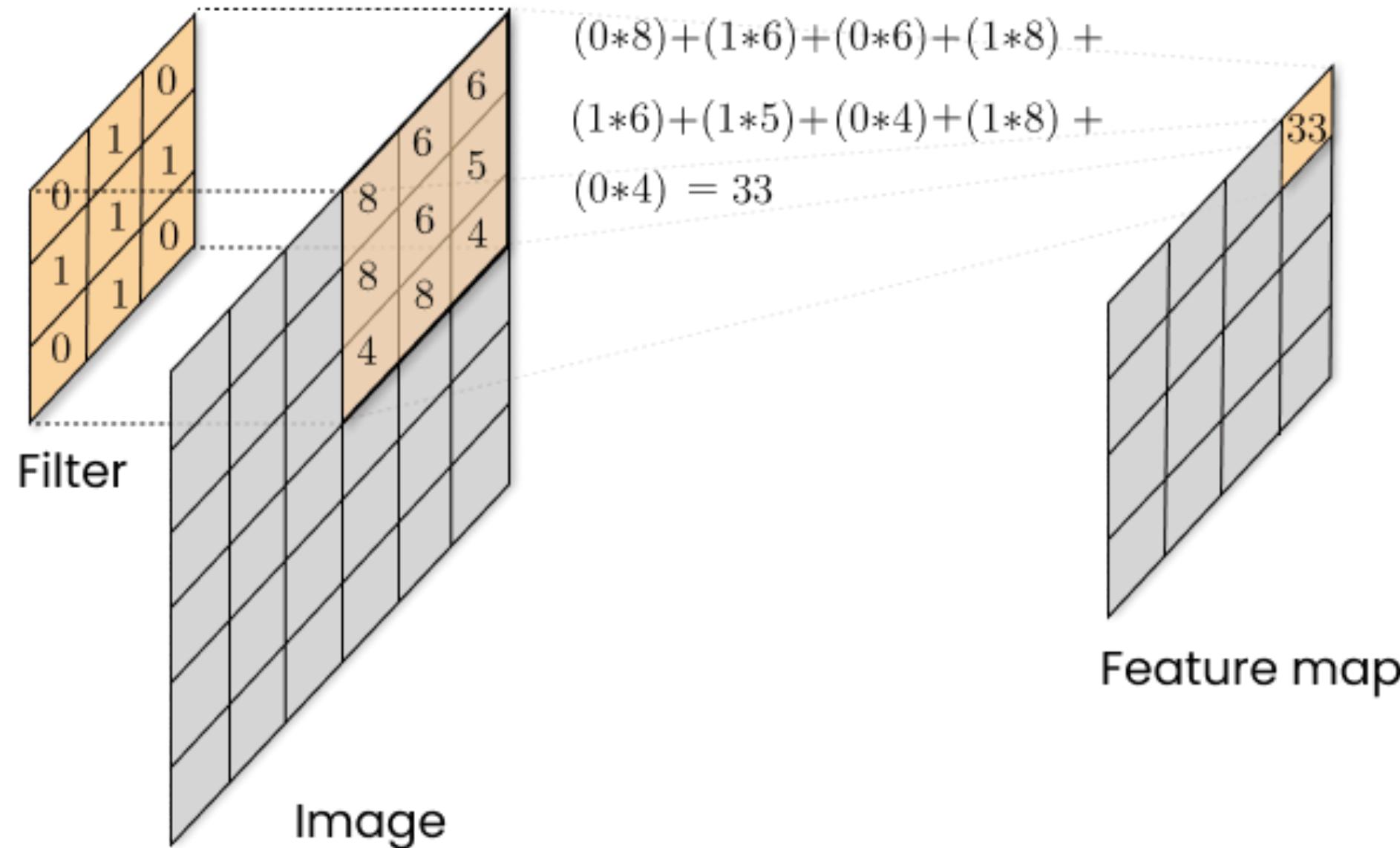
# CNN I



Machine Vision  
using Python (MVUP01)

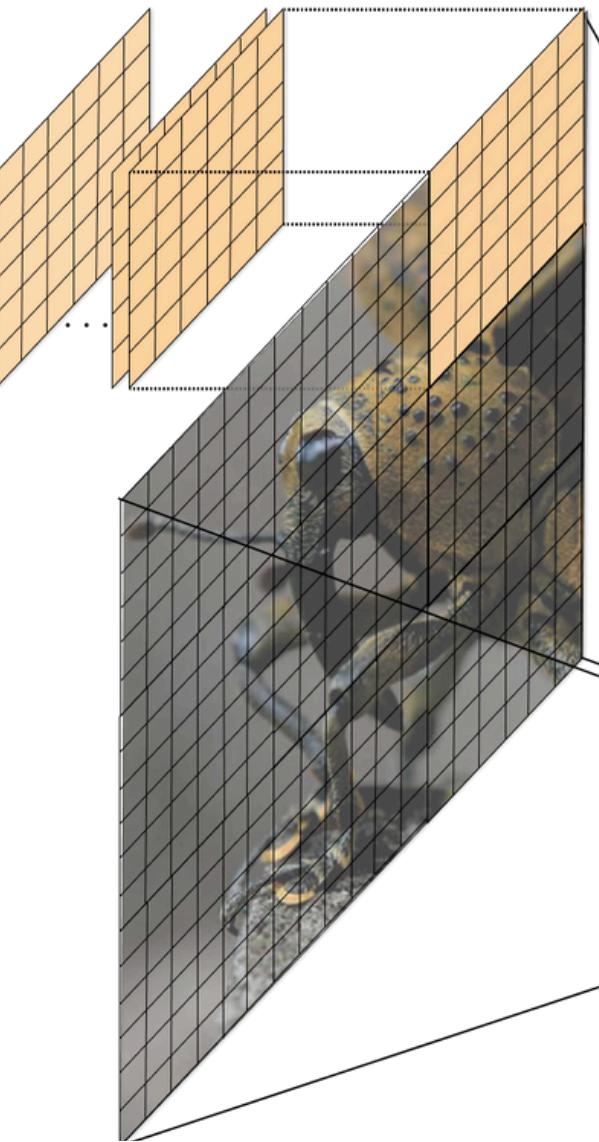


# CNN I



# CNN I

Convolution

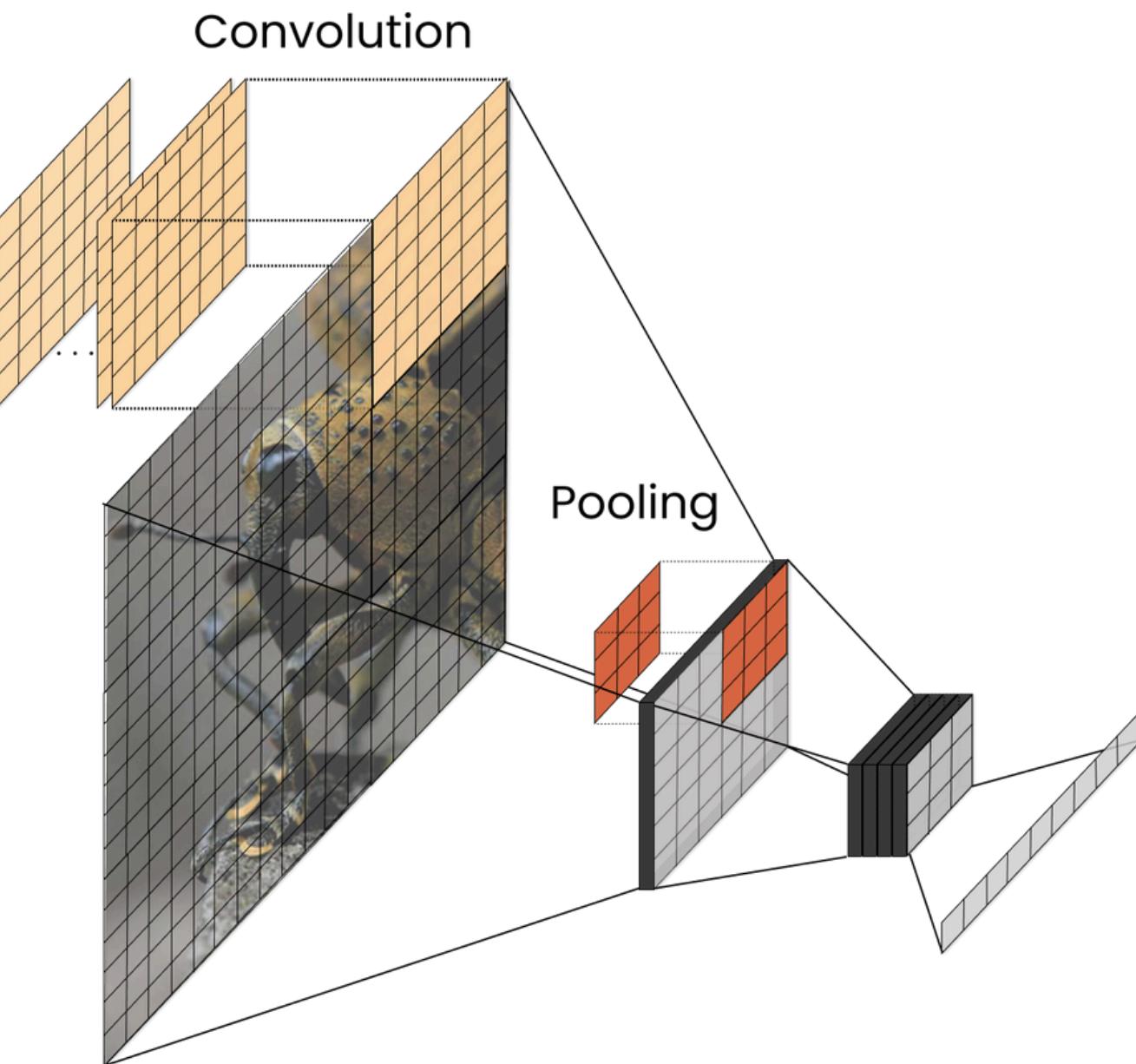


Machine Vision  
using Python (MVUP01)

R stats



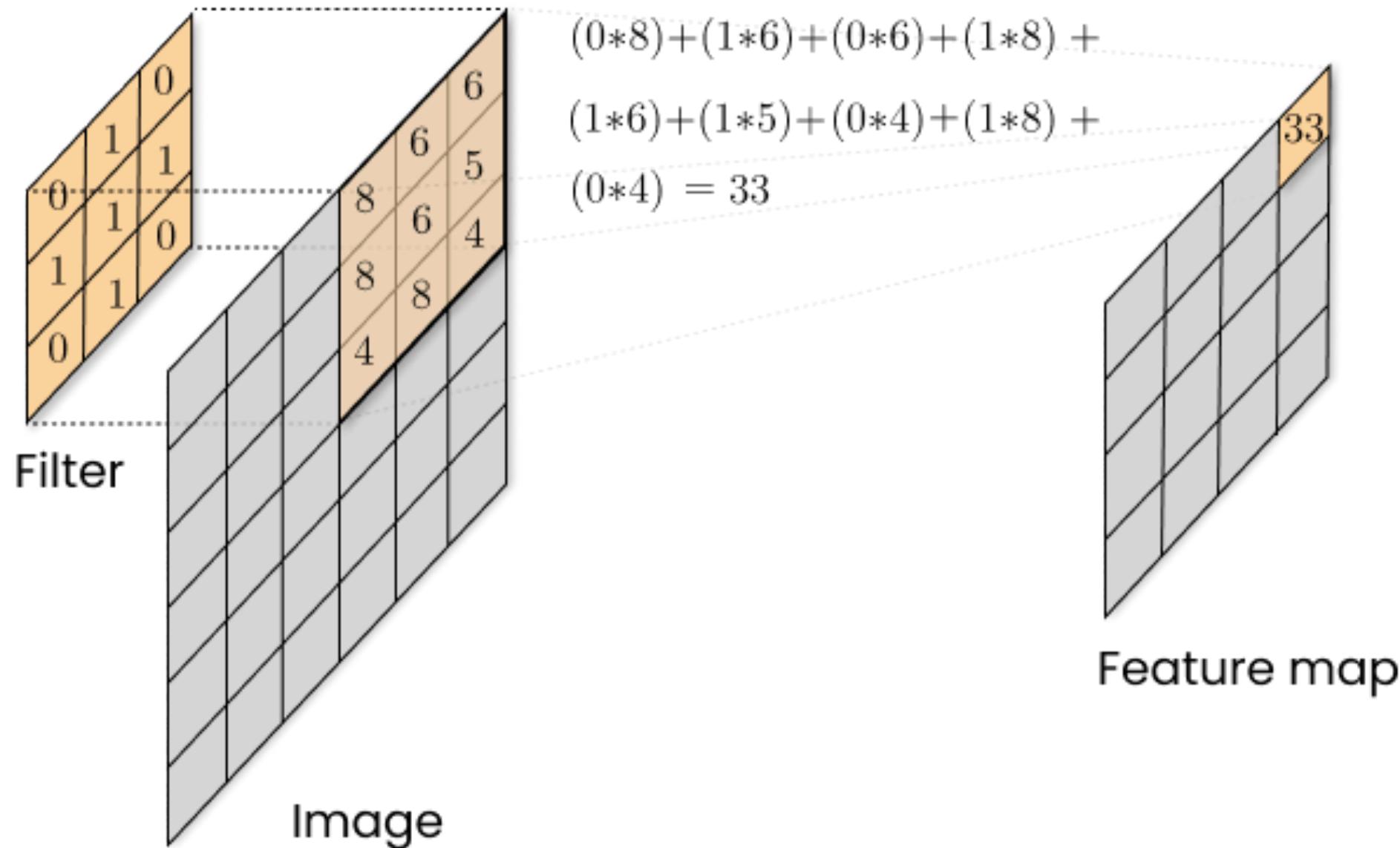
# CNN I



Machine Vision  
using Python (MVUP01)



# CNN I



# CNN I

## ConvNetJS CIFAR-10 demo

### Description

This demo trains a Convolutional Neural Network on the [CIFAR-10 dataset](#) in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used [this python script](#) to parse the [original files](#) (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

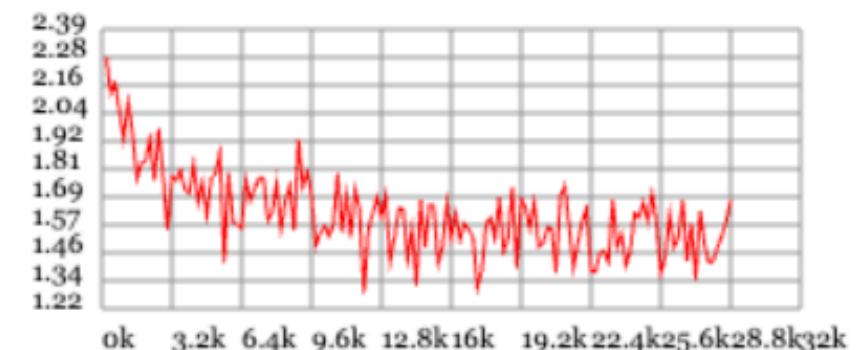
By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to [@karpathy](#).

### Training Stats

pause  
Forward time per example: 5ms  
Backprop time per example: 9ms  
Classification loss: 1.57914  
L2 Weight decay loss: 0.01882  
Training accuracy: 0.45  
Validation accuracy: 0.51  
Examples seen: 28957  
Learning rate:  change  
Momentum:  change  
Batch size:  change

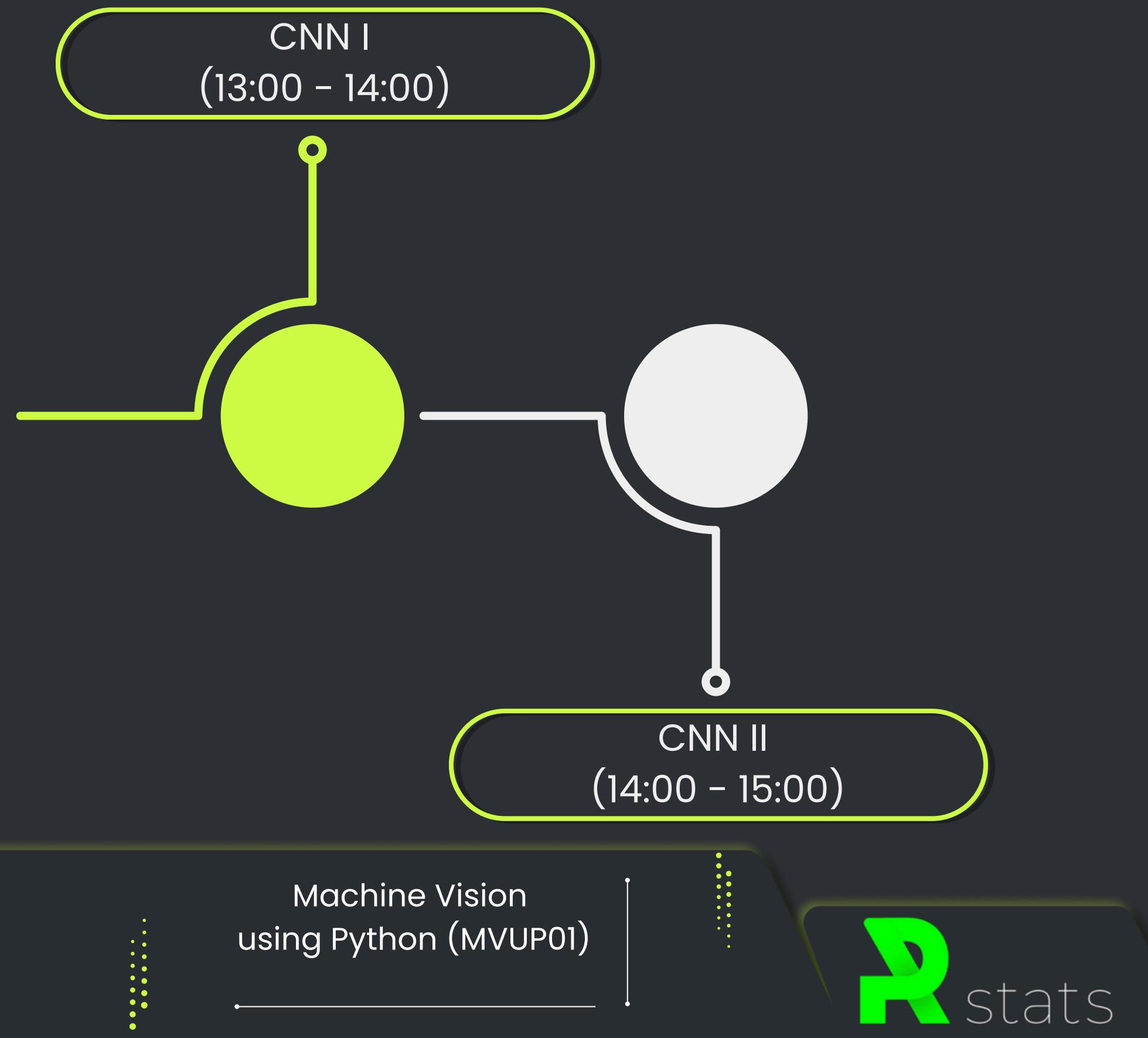
#### Loss:



Machine Vision  
using Python (MVUP01)



# Morning Session (9:30 - 12:00)



# CNN II

```
package i.ucd.be.before.solr;  
  
import ...  
  
public final class LocationUtils {  
  
    /**  
     * Parses Point from it's String representation.  
     * @param locationString - String that represents location, as 2 double values split with coma. Accepts space after/before coma  
     * @return org.springframework.data.solr.core.geo.Point instance  
     */  
    public static Point parseLocation(String locationString) {  
        Preconditions.checkNotNull(locationString, errorMessage: "Location String should not be null");  
        Preconditions.checkArgument(locationString.contains(","), errorMessage: "Location must be split with coma");  
        locationString = locationString.trim();  
  
        if (locationString.contains(" ")) {  
            locationString = locationString.replaceAll( regex: " ", replacement: "," );  
        }  
  
        if (locationString.contains(", ")) {  
            locationString = locationString.replaceAll( regex: ", ", replacement: "," );  
        }  
  
        String[] location = locationString.split( regex: "," );  
        Preconditions.checkNotNull( expression: location.length >= 2, errorMessage: "Location should consist at least 2 Double parameters" );  
        double lat = Double.parseDouble(location[0]);  
        double lon = Double.parseDouble(location[1]);  
  
        return new Point(lat, lon);  
    }  
}
```



```
@Autowired  
public DefaultCommunitySelService  
    communitySelRepository, CommunitySelDocumentPopulator, CommunityService communityService,  
    CommunitySelIndexerStrategy strategy  
{  
    this.communitySelRepository = communitySelRepository;  
    this.communitySelDocumentPopulator = communitySelDocumentPopulator;  
    this.communityService = communityService;  
    this.strategy = strategy;  
  
    @Override  
    public void index(Collection<Community> communities) {  
        Collection<CommunitySelDocument> documents = communities  
            .stream()  
            .map(community -> community.getCommunitySelDocument().convert(CommunitySelDocument::new))  
            .collect(Collectors.toList());  
        communitySelRepository.deleteAll();  
        communitySelRepository.saveAll(documents);  
    }  
    @Override  
    public Collection<Community> searchSelIndexQuery(CommunitySelIndexQuery query) {  
        List<CommunitySelDocument> documents = new ArrayList<>();  
        if (Collection<Community>.emptySet().equals(documents))  
            List<Community> retrievedCommunities = documents.stream().map(document -> communityService.getByName(document.getName())).collect(Collectors.toList());  
        else  
            LOG.warn("No places to index, input collection is empty.");  
        return communities;  
    }  
}
```

Machine Vision  
using Python (MVUP01)



# CNN II

Machine Vision  
using Python (MVUP01)



# CNN I

The CNN architecture contains more parameters to estimate than a DNN, mainly because of the convolution layers. This layer uses an operation called convolution that is defined by the equation below:

# CNN I

The CNN architecture contains more parameters to estimate than a DNN, mainly because of the convolution layers. This layer uses an operation called convolution that is defined by the equation below:

$$C_{r,q}^d = f \left( \sum_{m=1}^M \sum_{n=1}^N w_{m,n}^c x_{r+m-1, q+n-1} + \theta^c \right),$$

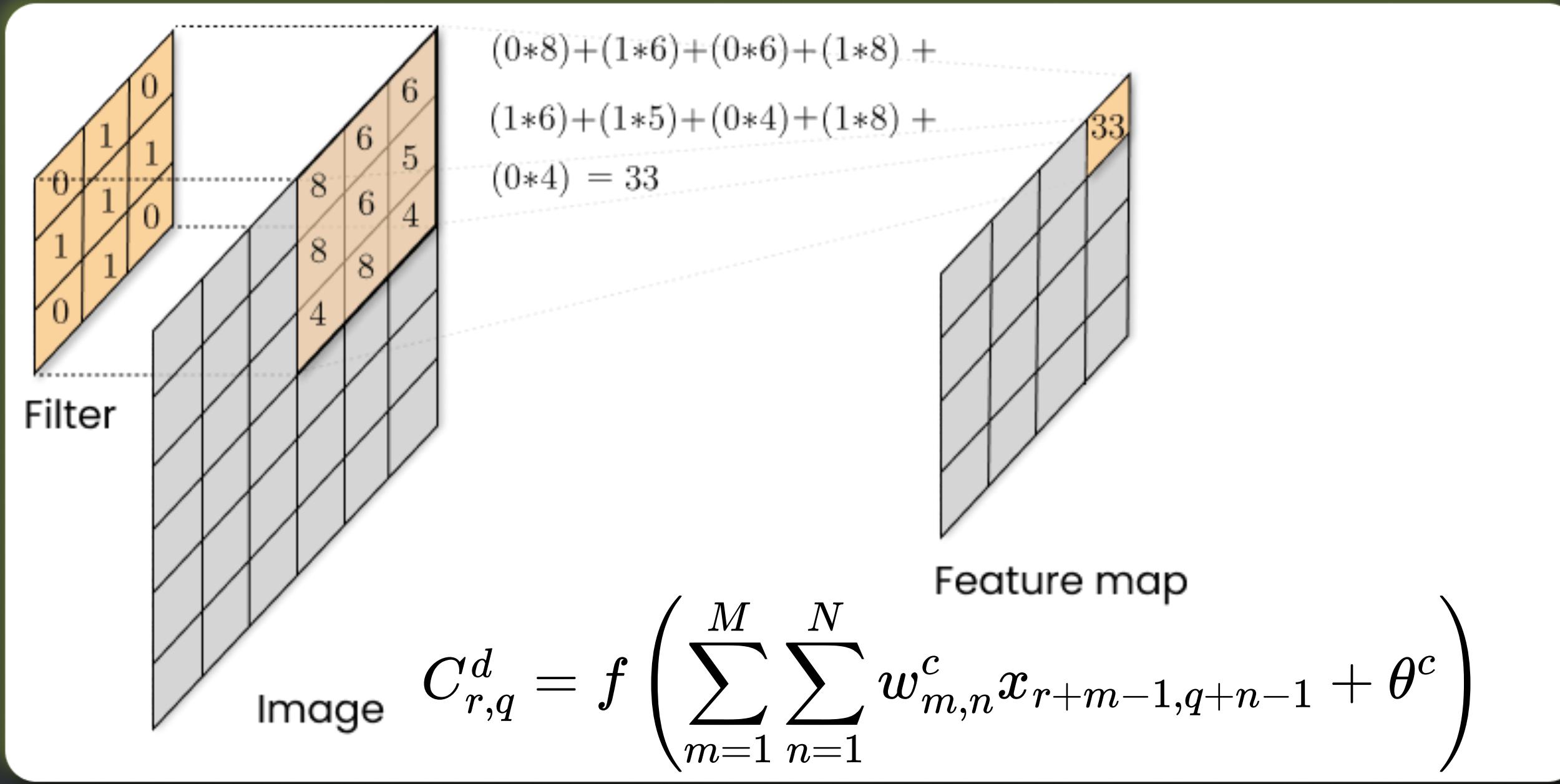
# CNN I

The CNN architecture contains more parameters to estimate than a DNN, mainly because of the convolution layers. This layer uses an operation called convolution that is defined by the equation below:

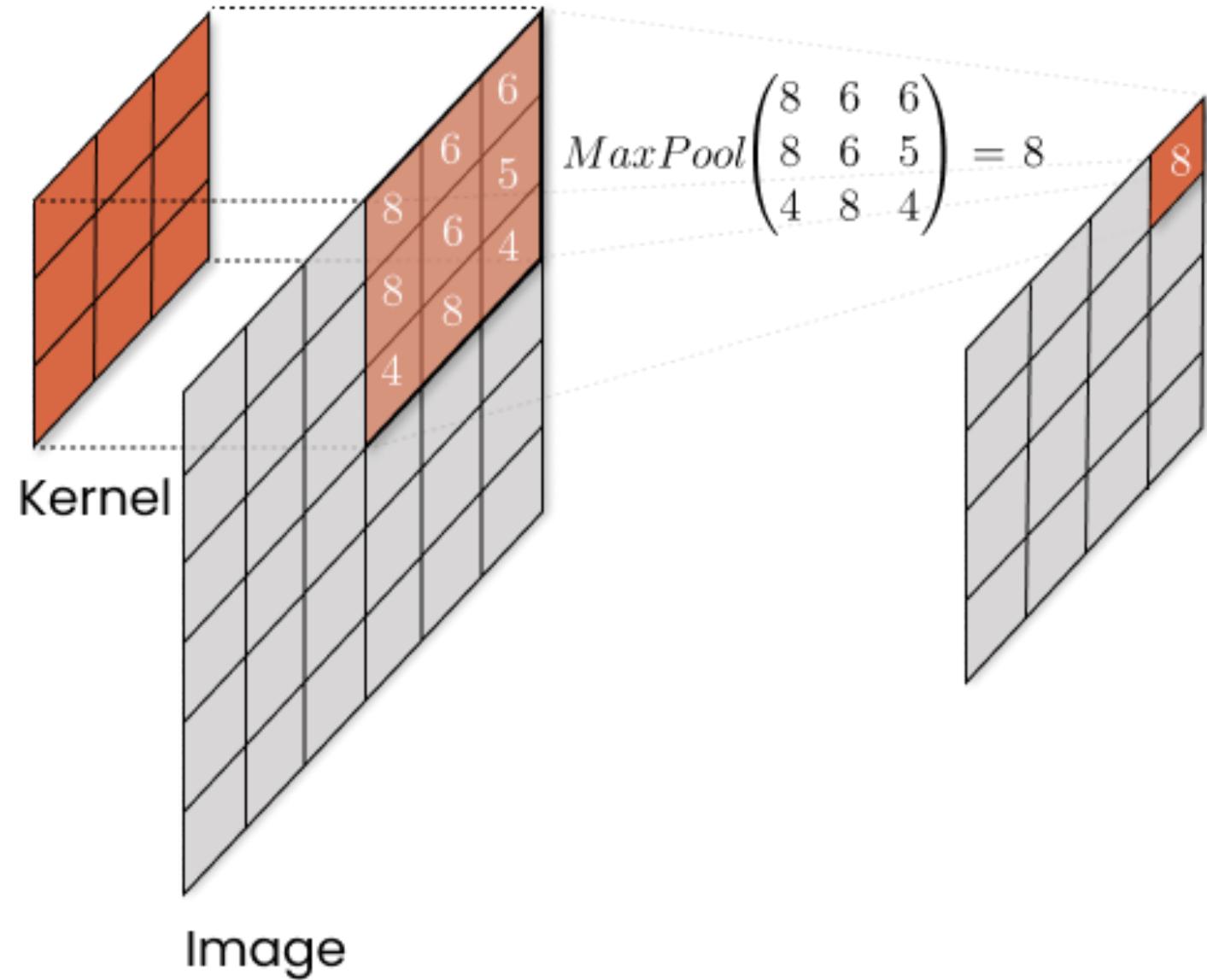
$$C_{r,q}^d = f \left( \sum_{m=1}^M \sum_{n=1}^N w_{m,n}^c x_{r+m-1, q+n-1} + \theta^c \right),$$

where  $x_{r,q}$  are pixels of a greyscale image  $\mathbf{X}$  of size  $R \times Q$ ,  $w_{m,n}^c$  and  $\theta^c$  are hyperparameters of a filter  $\mathbf{W}$  with size  $M \times N$ ,  $f$  can be any activation function, and  $(r, q)$  are the indices of the output, also known as a *feature map*. This operation can be repeated  $d$  times per convolution layer.

# CNN I



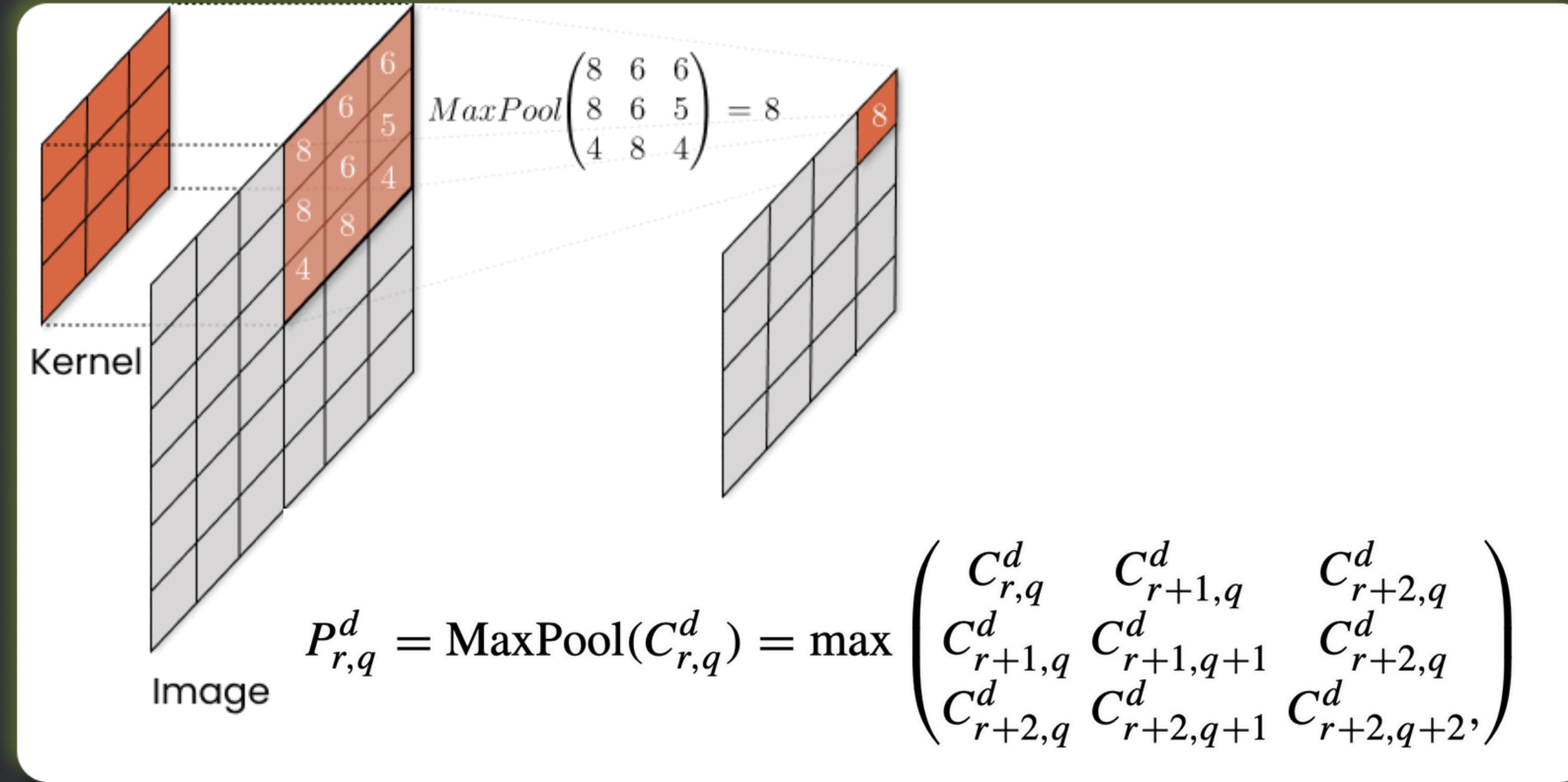
# CNN I



Machine Vision  
using Python (MVUP01)

R stats

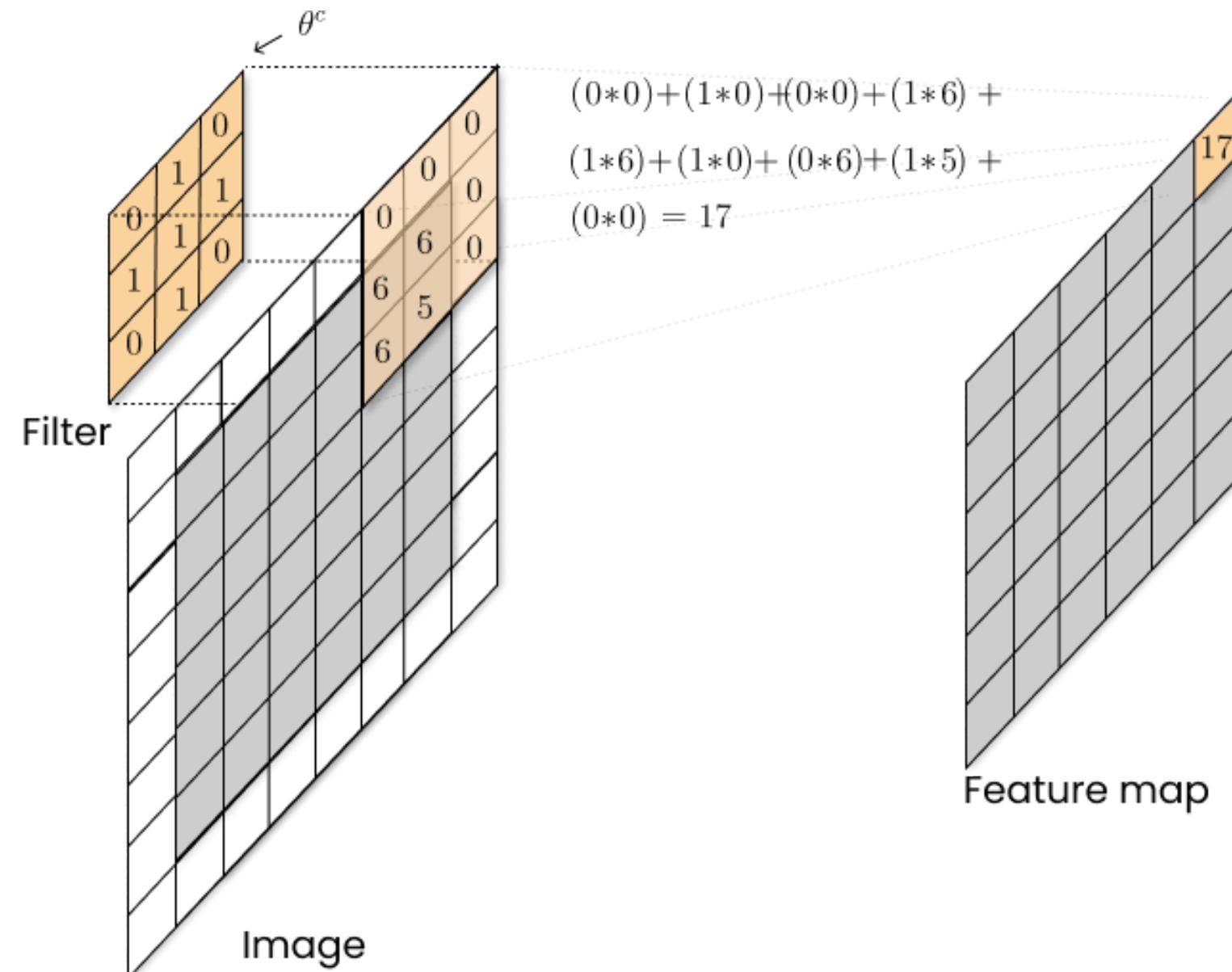
# CNN I



# CNN I

Typically since small filters are used in convolution operations, a few pixels may be lost. A solution for this problem is the padding approach, which consists of adding extra pixels around the boundary of the input image, increasing the effective size of the image [70]. These extra values are typically zero.

# CNN I



# CNN I

## ConvNetJS CIFAR-10 demo

### Description

This demo trains a Convolutional Neural Network on the [CIFAR-10 dataset](#) in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used [this python script](#) to parse the [original files](#) (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

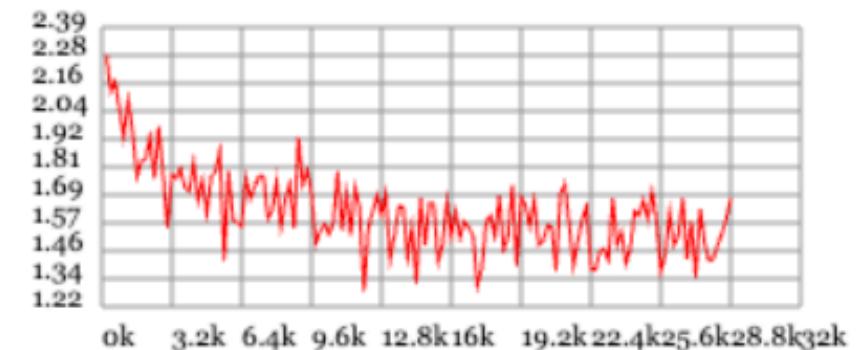
By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to [@karpathy](#).

### Training Stats

pause  
Forward time per example: 5ms  
Backprop time per example: 9ms  
Classification loss: 1.57914  
L2 Weight decay loss: 0.01882  
Training accuracy: 0.45  
Validation accuracy: 0.51  
Examples seen: 28957  
Learning rate:  change  
Momentum:  change  
Batch size:  change

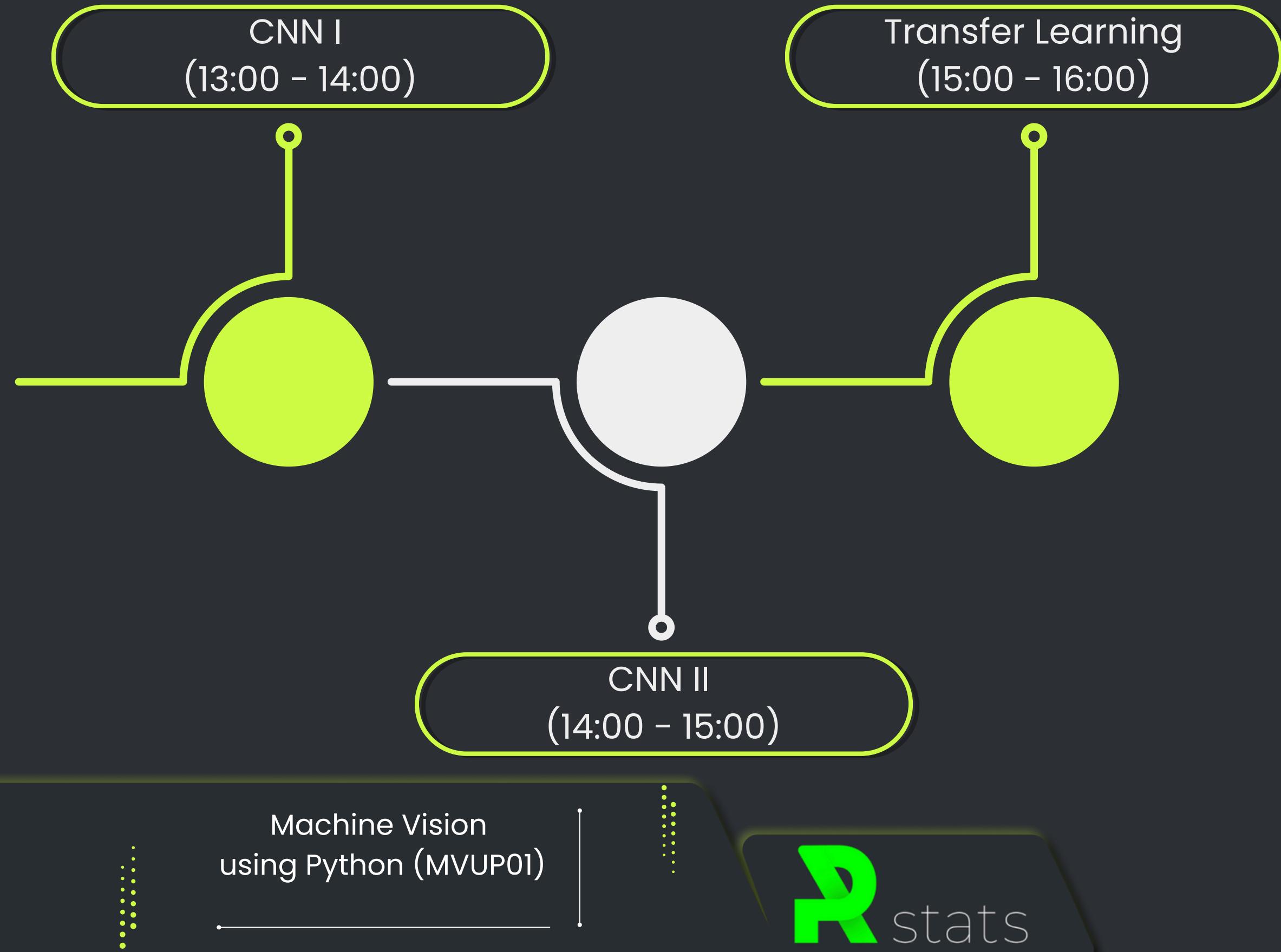
#### Loss:



Machine Vision  
using Python (MVUP01)



# Morning Session (9:30 - 12:00)

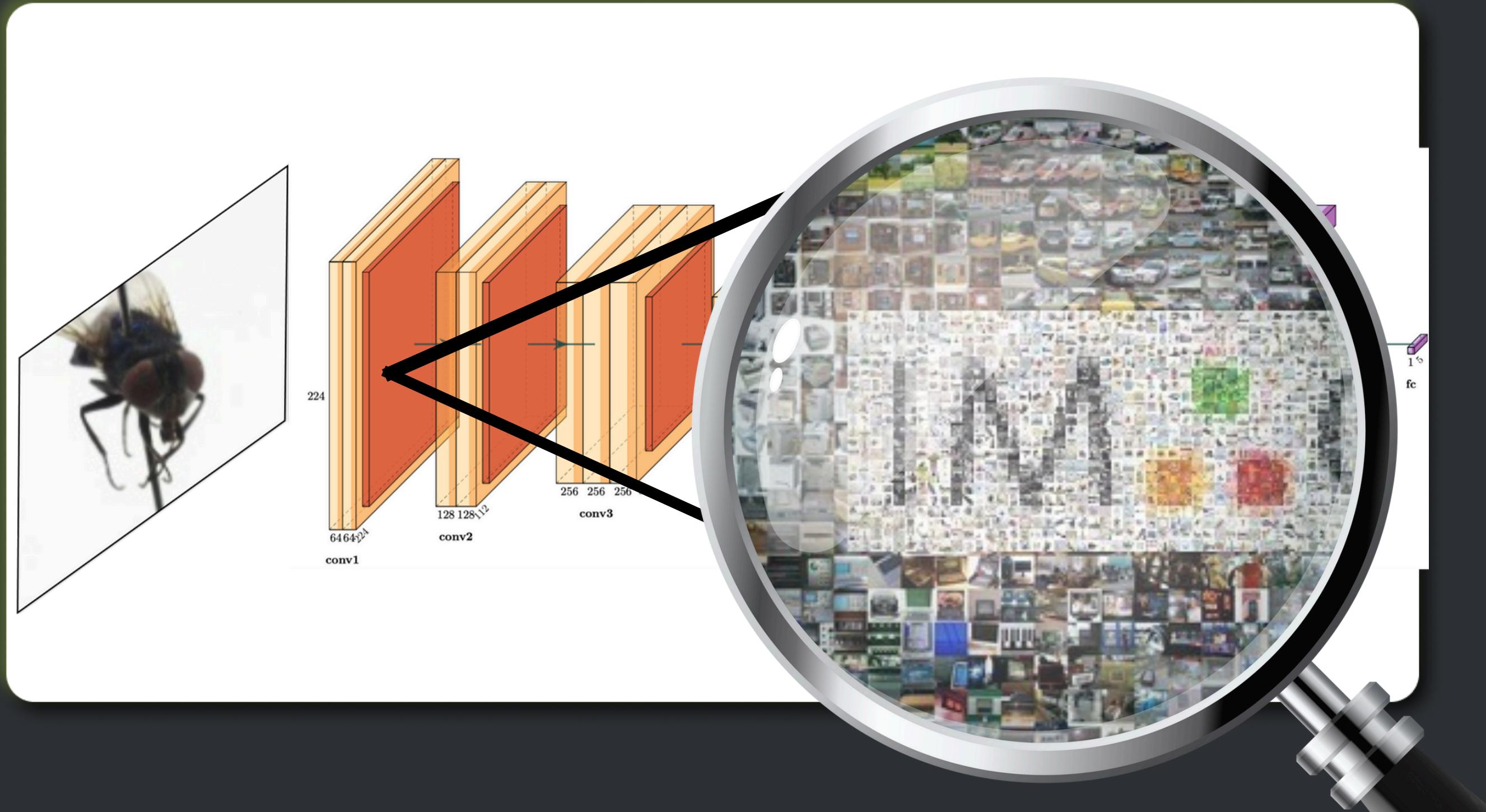


# Transfer Learning

Machine Vision  
using Python (MVUP01)



# CNN II



Machine Vision  
using Python (MVUP01)



# CNN II

3058 dataset results for **Images** X

Search for datasets  🔍

Best match

**Filter by Modality** [\(clear\)](#)

Modality	Count
Images	2931
Texts	2931
Videos	959
Audio	457
Medical	376
3D	358

---

**CIFAR-10 (Canadian Institute for Advanced Research, 10 classes)**  
The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is di...  
15,538 PAPERS • 108 BENCHMARKS

---

**ImageNet**  
The ImageNet dataset contains 14,197,122 annotated images according to the WordNet hierarchy. Since 2010 the dataset is used in the ImageNet Large Scale Visual Recognition...  
14,860 PAPERS • 52 BENCHMARKS

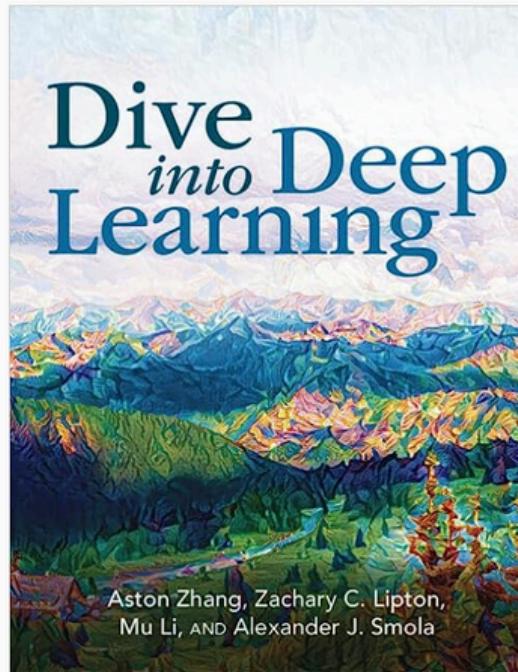
---

**MS COCO (Microsoft Common Objects in Context)**  
The MS COCO (Microsoft Common Objects in Context) dataset is a large-scale object detection, segmentation, key-point detection, and captioning dataset. The dataset consists of...  
11,420 PAPERS • 96 BENCHMARKS

---

Machine Vision  
using Python (MVUP01)





## Dive into Deep Learning

Interactive deep learning book with code, math, and discussions

Implemented with PyTorch, NumPy/MXNet, JAX, and TensorFlow

Adopted at 500 universities from 70 countries



Star

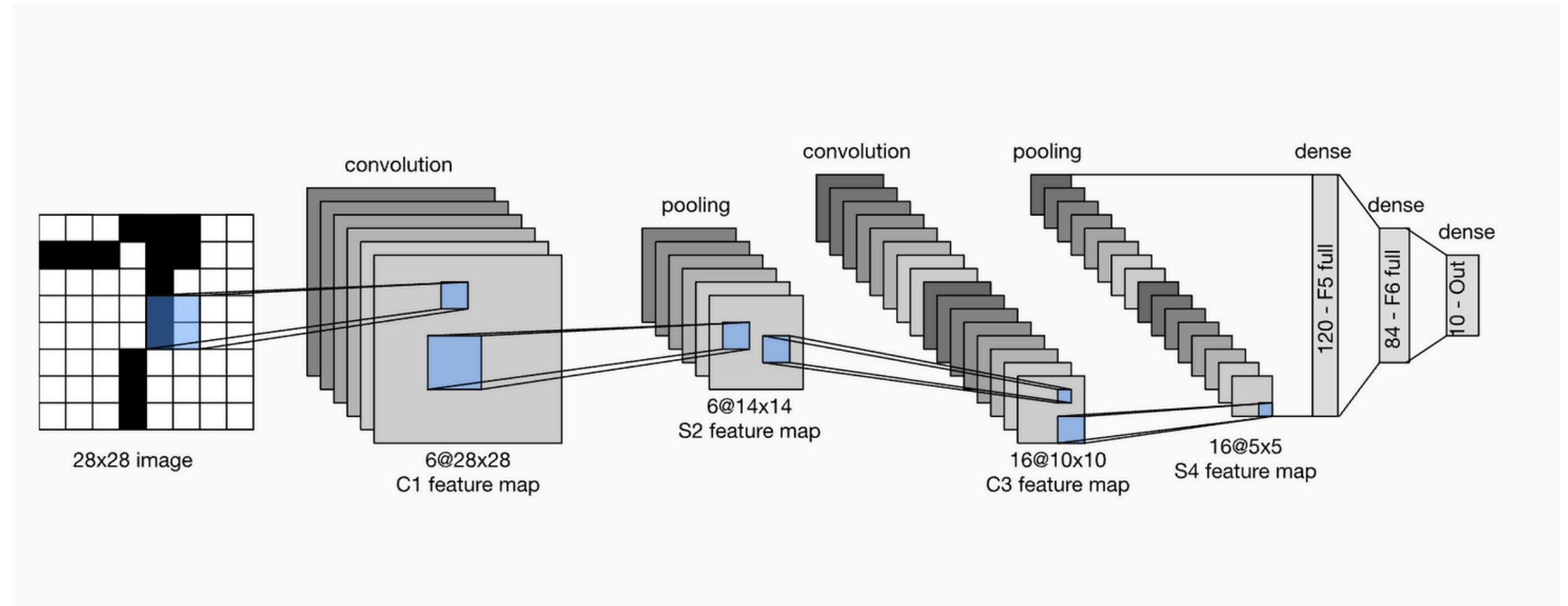
24,823

Machine Vision  
using Python (MVUP01)



# CNN I

## LeNet

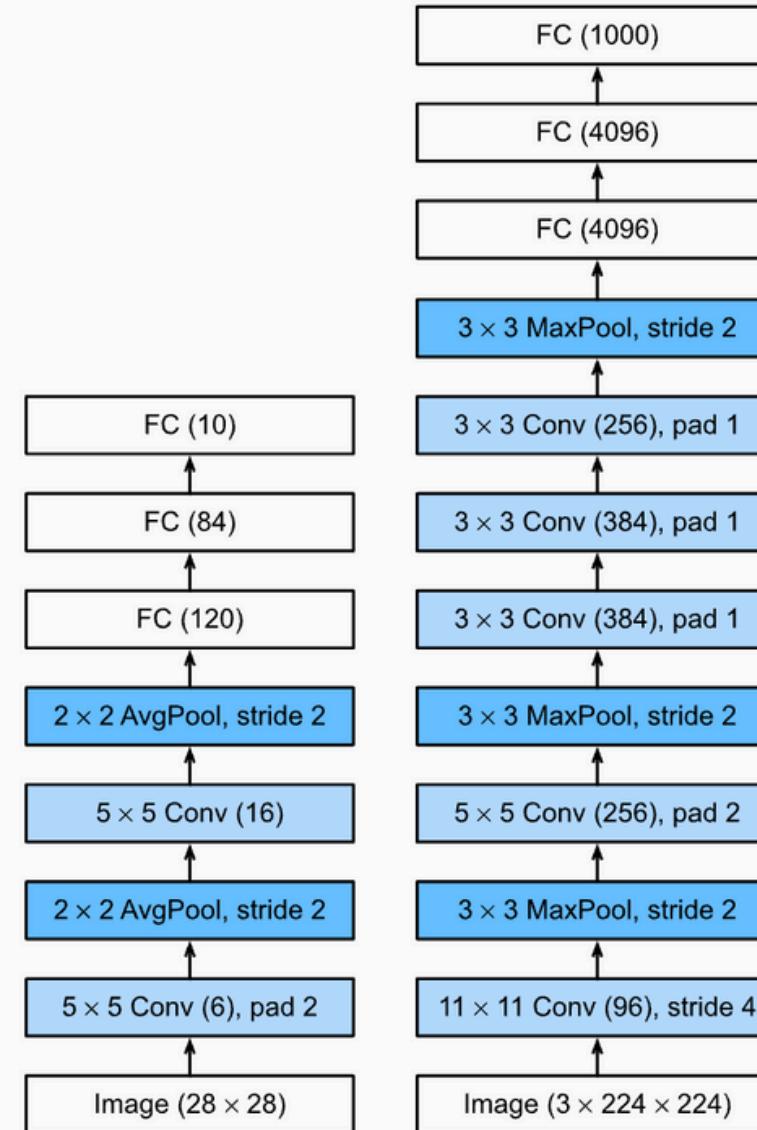


Machine Vision  
using Python (MVUP01)



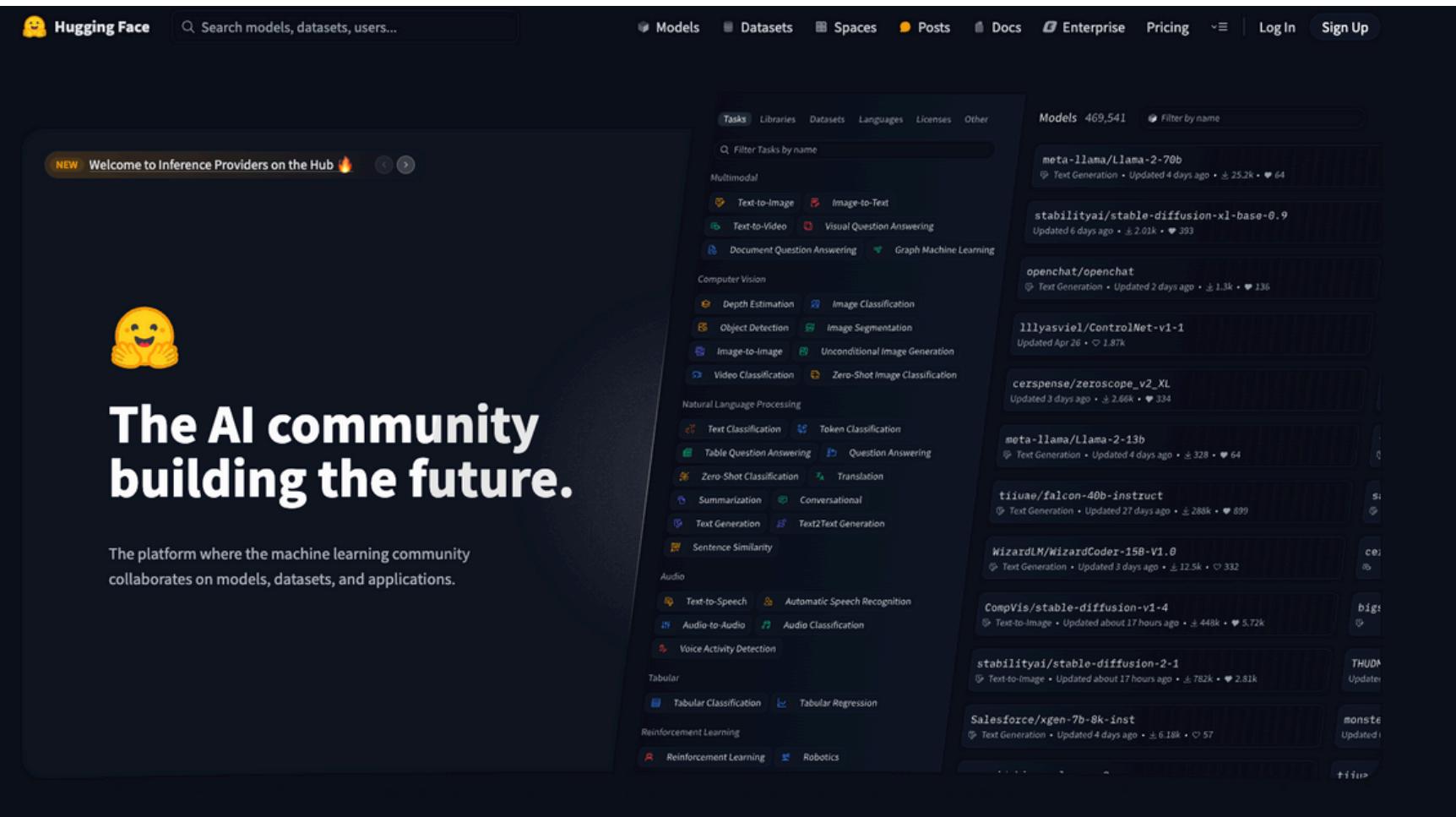
# CNN I

## LeNet VS AlexNet



# CNN I

## huggingface.co

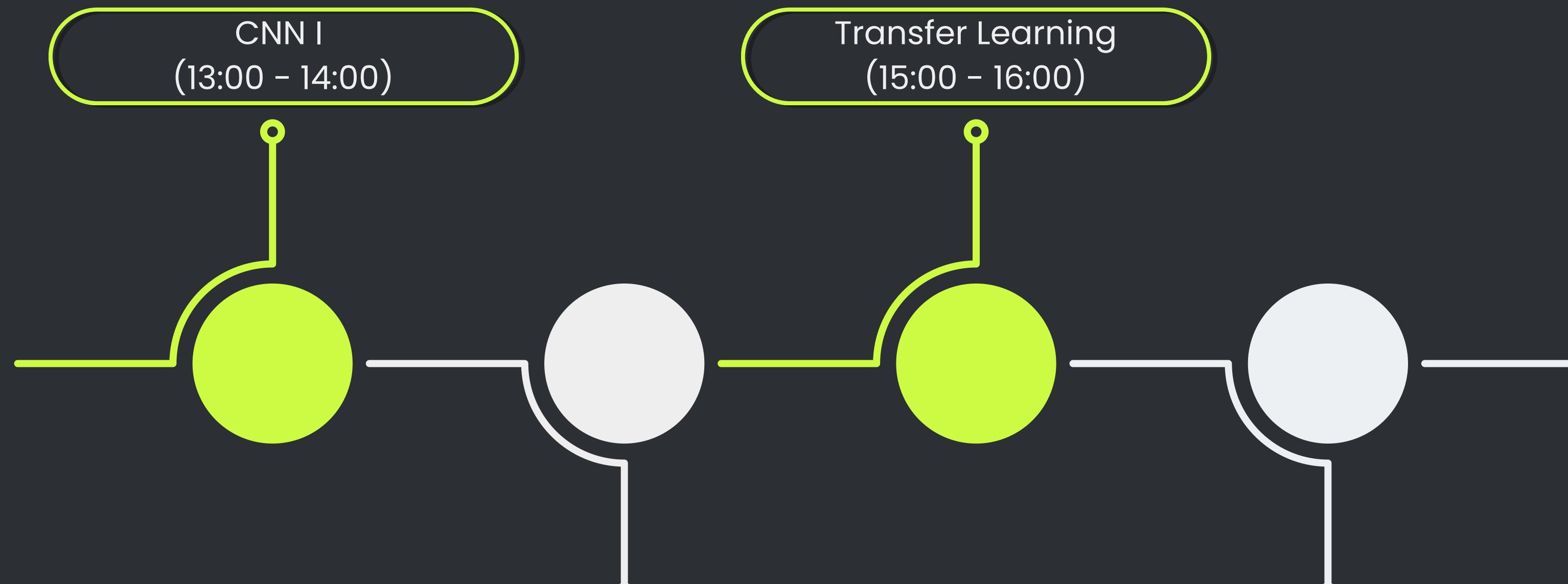


Machine Vision  
using Python (MVUP01)

R stats



# Afternoon Session (13:00 - 17:30)



Machine Vision  
using Python (MVUP01)



# Putting It All Together

```
// Types can be a map of types/handlers
if ( typeof types === "object" ) {
  // types-Object, selector, data
  if ( typeof selector !== "string" ) {
    data = data || selector;
    selector = undefined;
  }
  for ( type in types ) {
    on( elem, type, selector, data, types[ type ], one );
  }
  return elem;
}
if ( data == null && fn == null ) {
```