Convolutional Neural Networks (CNNs): Basic Structure

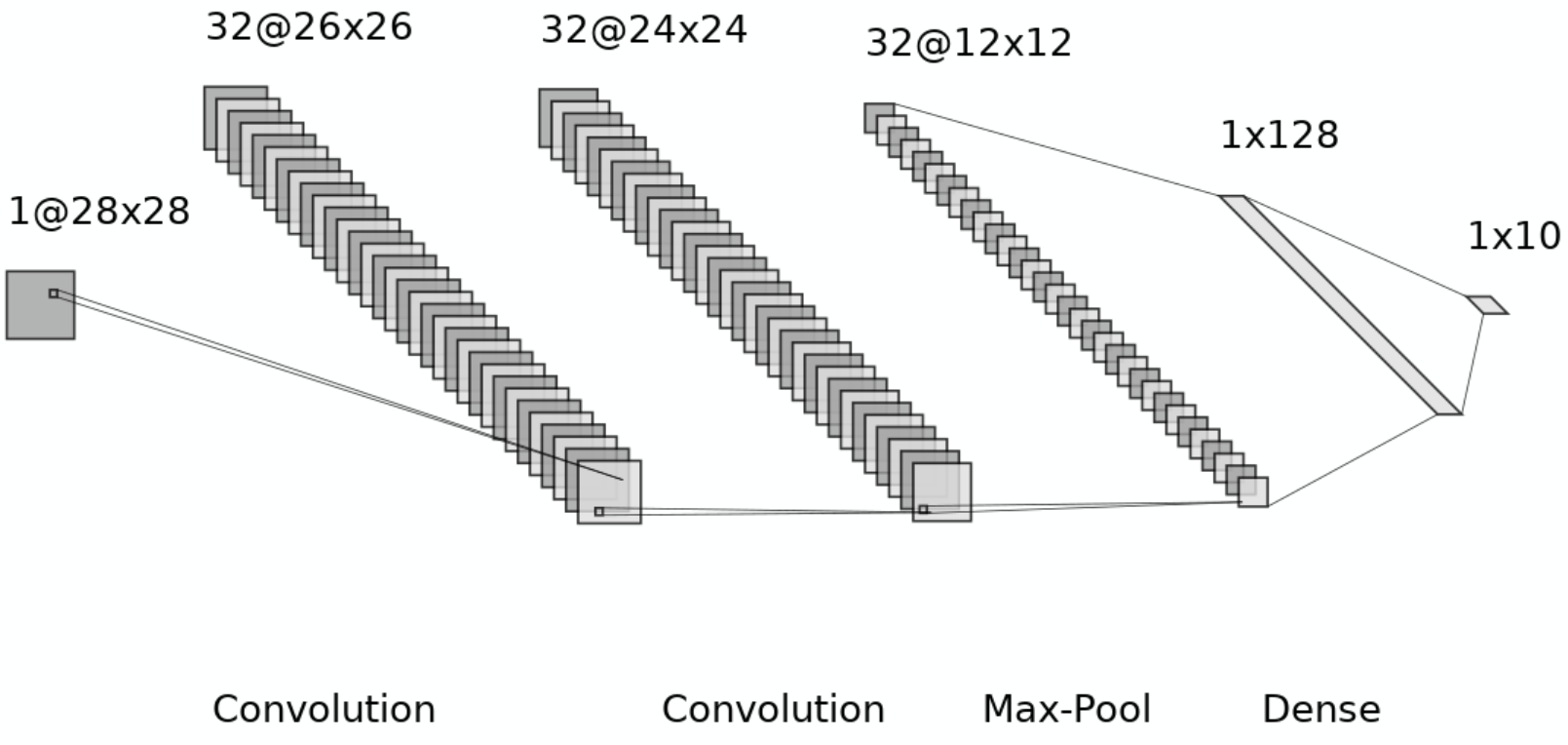
Laura E. Boucheron

College of Engineering | Klipsch School of Electrical and Computer Engineering

BE BOLD. Shape the Future.

A Visualization of our MNIST network

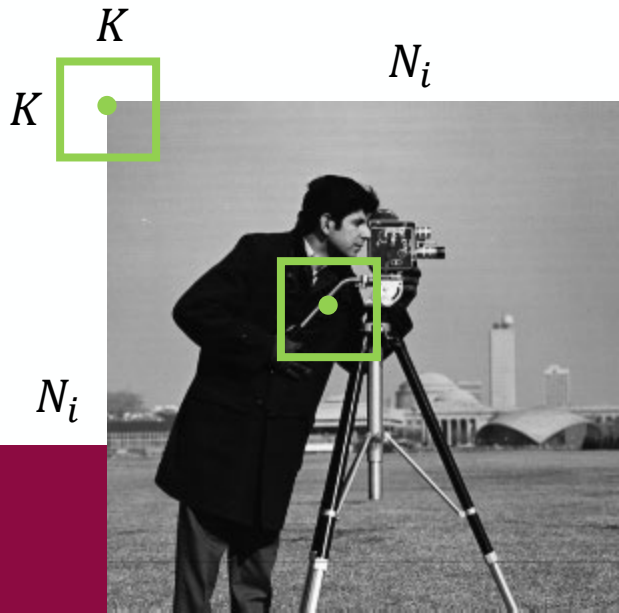
- The `Sequential` model allows you to stack layers in a linear fashion



Types of Layers

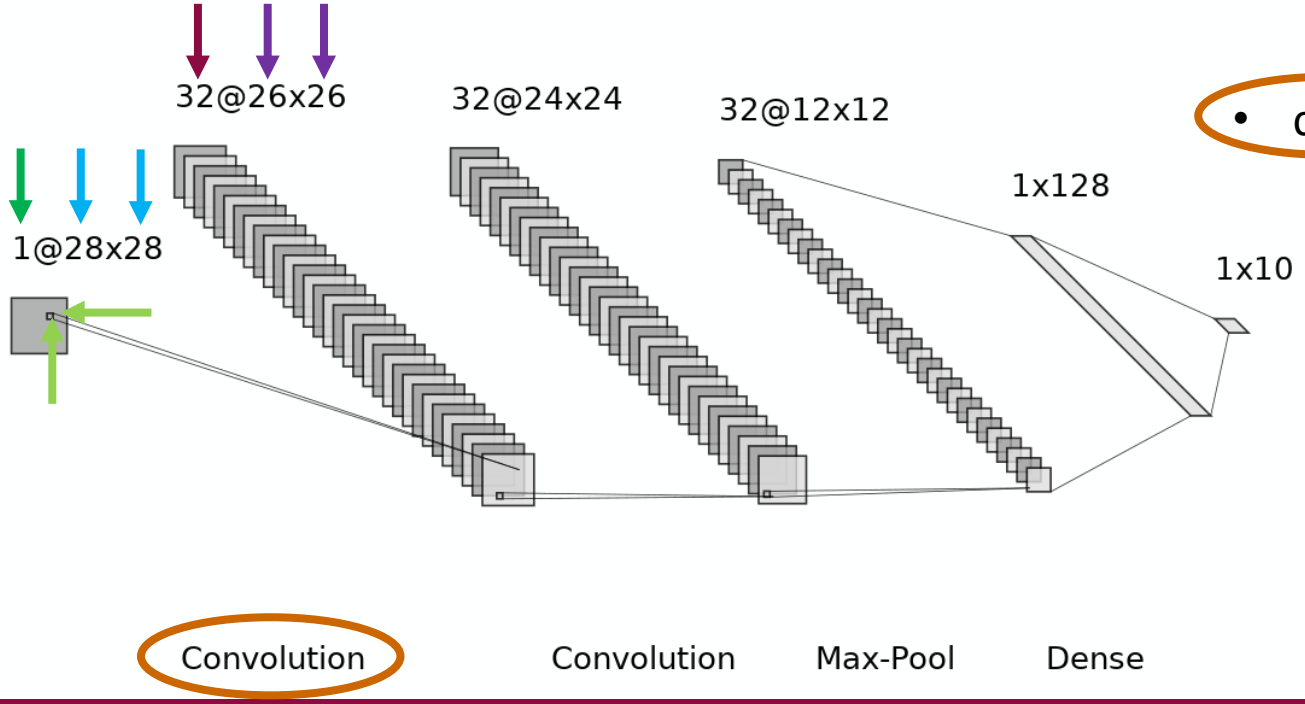
Convolutional Layers (`Convolution2D`)

- Learn some M_{conv} $K \times K \times C$ filters (and M_{conv} biases)
  - M_{conv} is the total number of filters in the current layer
  - K is the kernel size
  - C is the number of channels over which the filter operates
- Input is some $N_i \times N_i \times C$ tensor (think multidimensional image)
  - N_i is the spatial dimension
  - C is the number of channels
- Output are some M_{conv} $N_o \times N_o$ filtered images, also commonly called "feature maps" or "activations"
  - N_o is the spatial dimension
    - Commonly, $N_o = N_i - (K - 1)$since the filtered output at the boundaries of the image is less reliable than at the center



Convolutional Layers (Convolution2D)

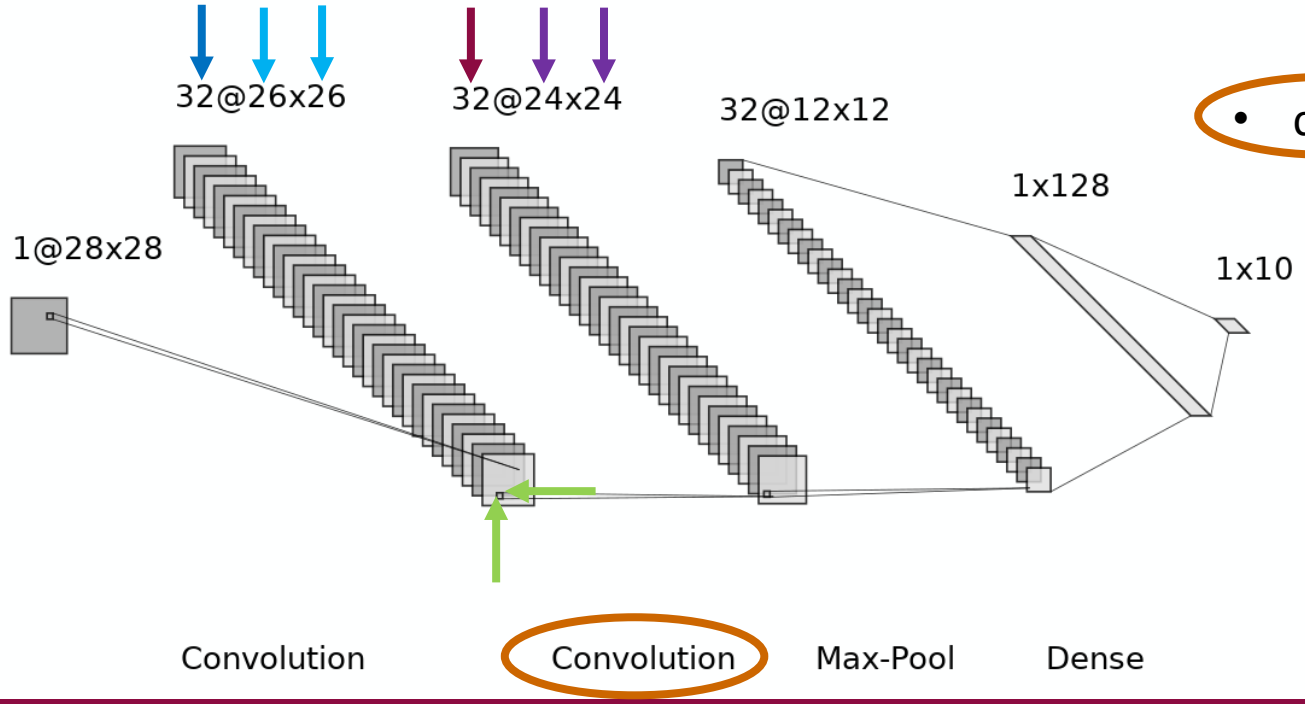
		conv1
• M_{conv}	number of filters in the current layer	32
• K	kernel size	3
• C	number of channels over which the filter operates	1
• N_i	spatial dimension of the input	28
• N_o	spatial dimension of the output	26



- conv1
  - Learn 32, 3x3x1 filters
  - Input is the 28x28x1 image
  - Output are the 32 26x26 activations

Convolutional Layers (Convolution2D)

		conv2
• M_{conv}	number of filters in the current layer	32
• K	kernel size	3
• C	number of channels over which the filter operates	32
• N_i	spatial dimension of the input	26
• N_o	spatial dimension of the output	24

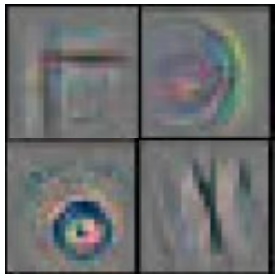


- conv1
  - Learn 32, 3x3x32 filters
  - Input is the 26x26x32 activations
  - Output are 32 24x24 activations

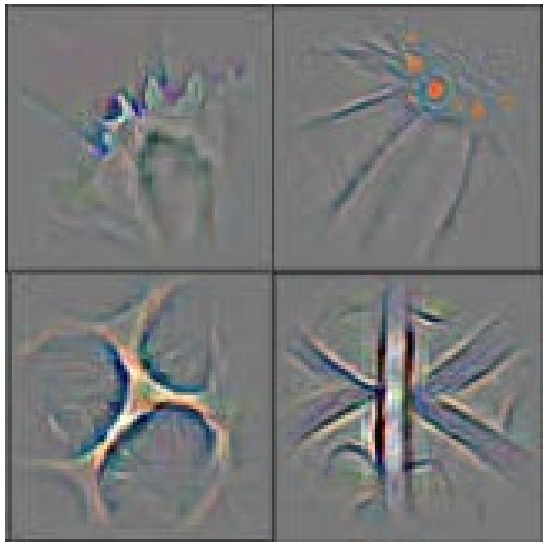
Convolutional Layers (`Convolution2D`)

- Hierarchical representation of image content

- First layer often converges to filters that highlight oriented edges



- Second layer can combine oriented edges and represent more complex structures such as corners, circles, or other shapes



- Third layer can combine those shapes and learn to represent even more abstract structures

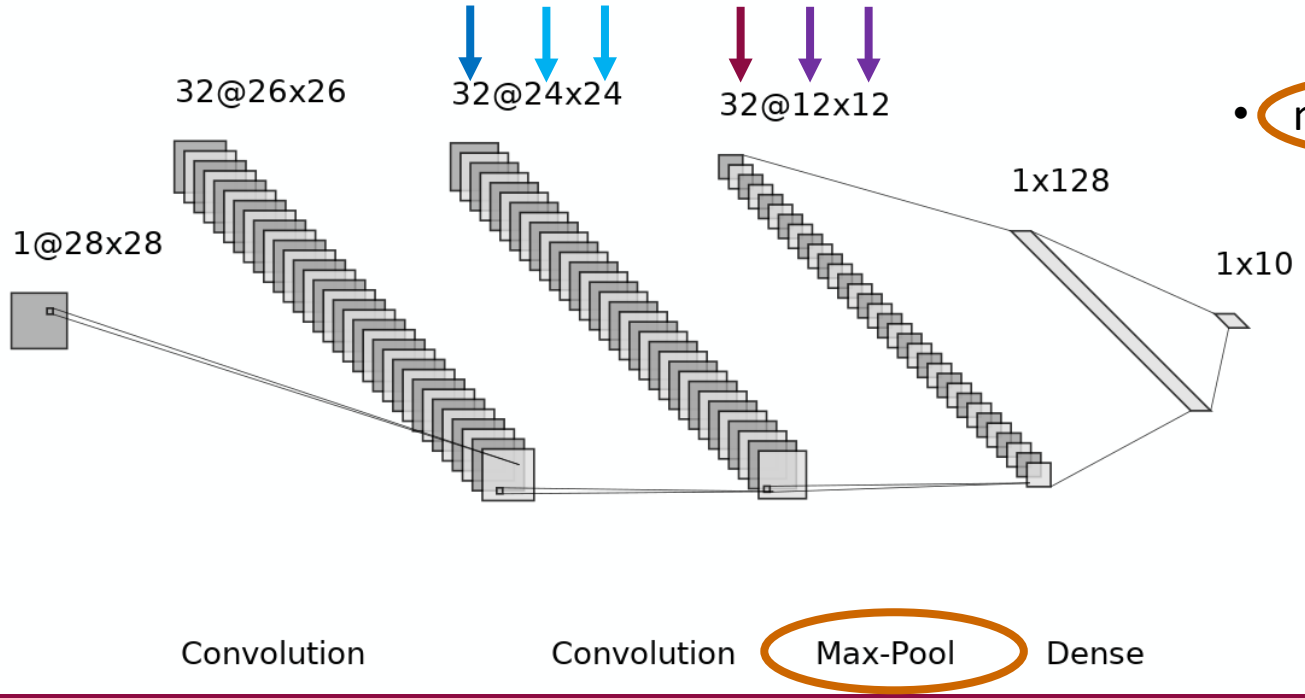
Figures from: Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833).

Pooling Layers (`MaxPooling2D`)

- Reduce the spatial resolution via subsampling, usually after a convolutional layer
  - Pool size P
- Input is some $N_i \times N_i \times C$ image
  - N_i is the spatial dimension of the input
  - C is the number of channels
- Output are some $C \frac{N_i}{P} \times \frac{N_i}{P}$ subsampled activations

Pooling Layers (`MaxPooling2D`)

		maxpool1
• P	pool size	2
• N_i	spatial dimension of the input	24
• C	number of channels	32

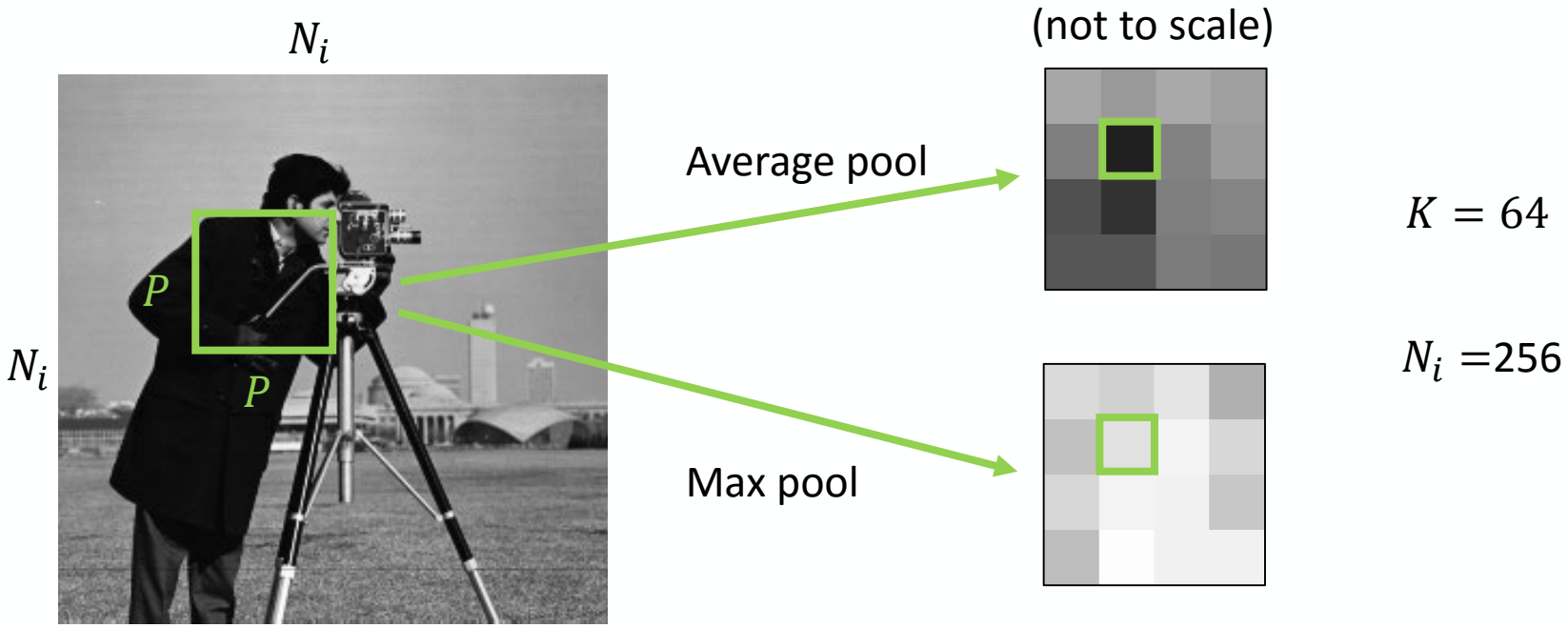


- maxpool1
  - Learn nothing
  - Input is the 24x24x32 activations
  - Output are the 32 12x12 subsampled activations

$$\frac{N_i}{P} = \frac{24}{2} = 12$$

Pooling Layers (`MaxPooling2D`)

- Reduce the spatial resolution via subsampling
  - Reduces the computational complexity (fewer pixels to filter)
  - Contributes to the scale invariance and hierarchical nature of the network
- Multiple forms of pooling
  - **Max pooling** (common): take maximum value within $P \times P$ pooling window
  - Average pooling: take average value within $P \times P$ pooling window
  - Others



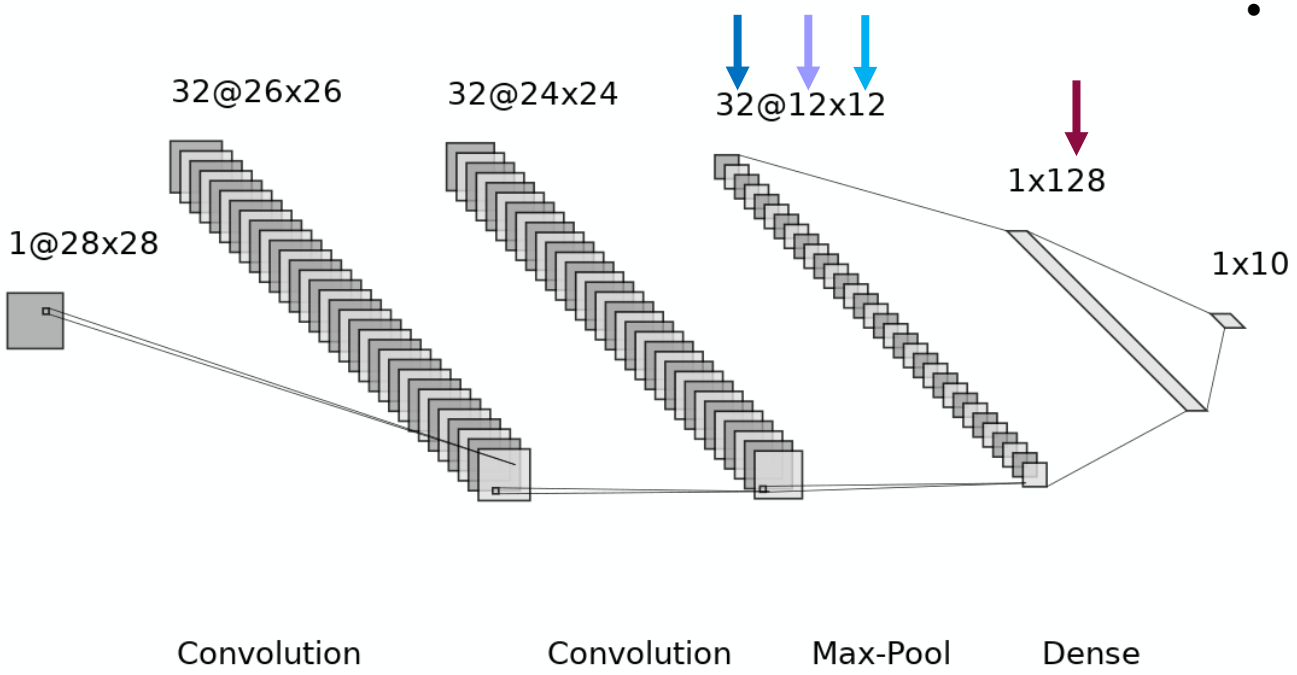
Fully Connected (Dense)

- Learn some M_{FC} nodes ($M_{FC} \cdot N_1 \cdot N_2 \cdot N_3$ weights and M_{FC} biases) associated with a standard fully connected 1D neural network
  - M_{FC} is the number of nodes in the current fully connected layer
  - N_1, N_2, and N_3 are the dimensions of the input
- Input is some $1 \times N_1 \cdot N_2 \cdot N_3$ tensor of activations from the previous layer
  - The `Flatten` function "flattens" the $N_1 \times N_2 \times N_3$ tensor output from the previous layer to a $1 \times N_1 \cdot N_2 \cdot N_3$ tensor
- Output are some M_{FC} activations

Fully Connected

FC1

- M_{FC} number of nodes in the current fully connected layer 128
- N_1, N_2, N_3 dimensions of the input 12, 12, 32

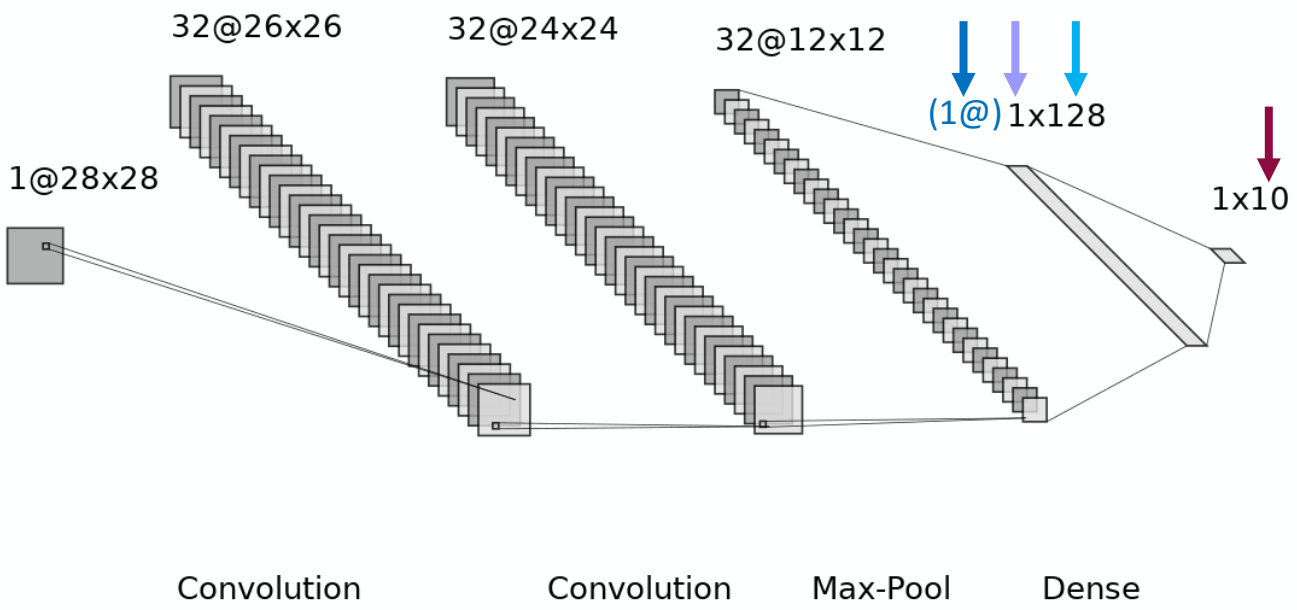


- FC1
  - Learn 128 nodes
  - Input are the 12x12x32 activations
  - Output are the 128 activations

Fully Connected

FC1

- M_{FC} number of nodes in the current fully connected layer 10
- N_1, N_2, N_3 dimensions of the input 1, 128, 1

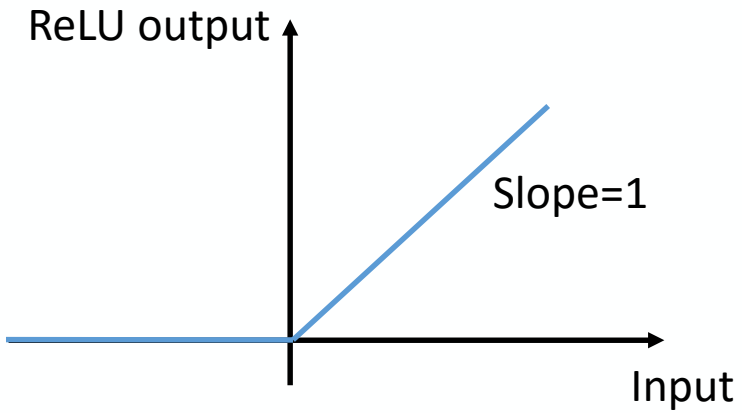


- FC1
  - Learn 10 nodes
  - Input are the 1x128x1 activations
  - Output are the 128 activations

Activations

Activations

- Included as the `activation` option in the `Convolution2D` and `Dense` layers
- Common activation for convolutional layers is the rectified linear unit (ReLU), `activation='relu'`
  - Clips negative values to 0
  - Identity for positive values

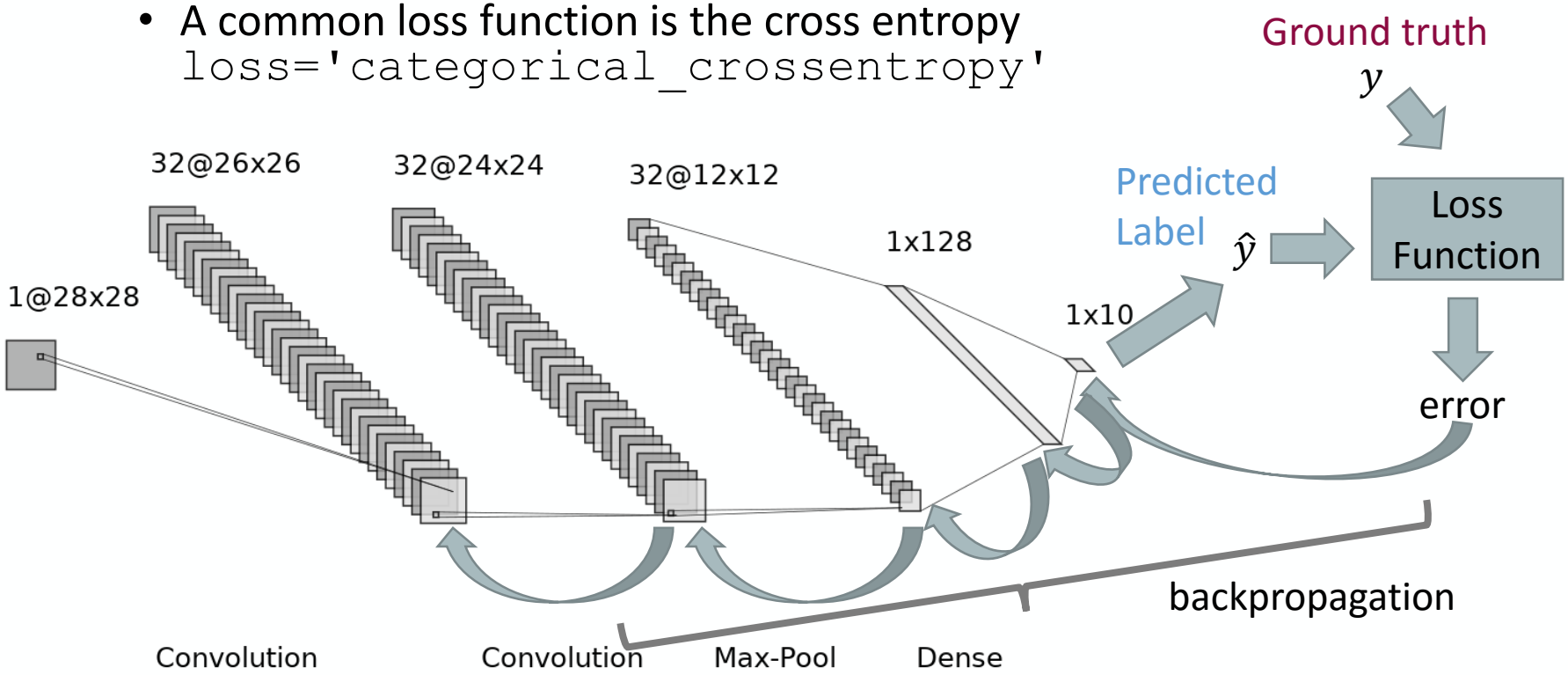


- Common activation for output of classifier network is the softmax, `activation='softmax'`
  - Normalizes the input to probabilities of class membership

Training Parameters

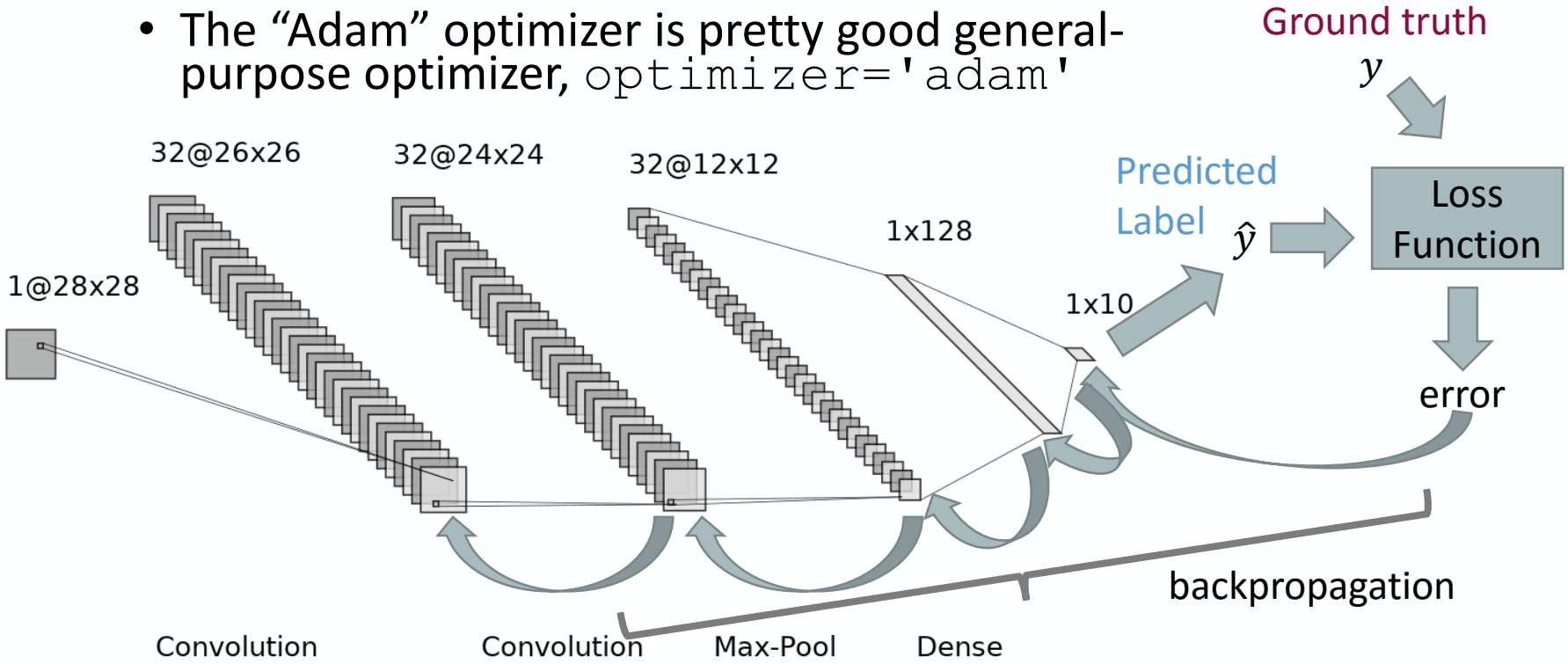
Loss Functions

- The loss function is THE metric that the NETWORK USES to train
- The loss function should somehow compare the current predicted labels to the ground truth labels
  - Should have good computational characteristics (e.g., doesn't overflow)
  - A common loss function is the cross entropy
    `loss='categorical_crossentropy'`



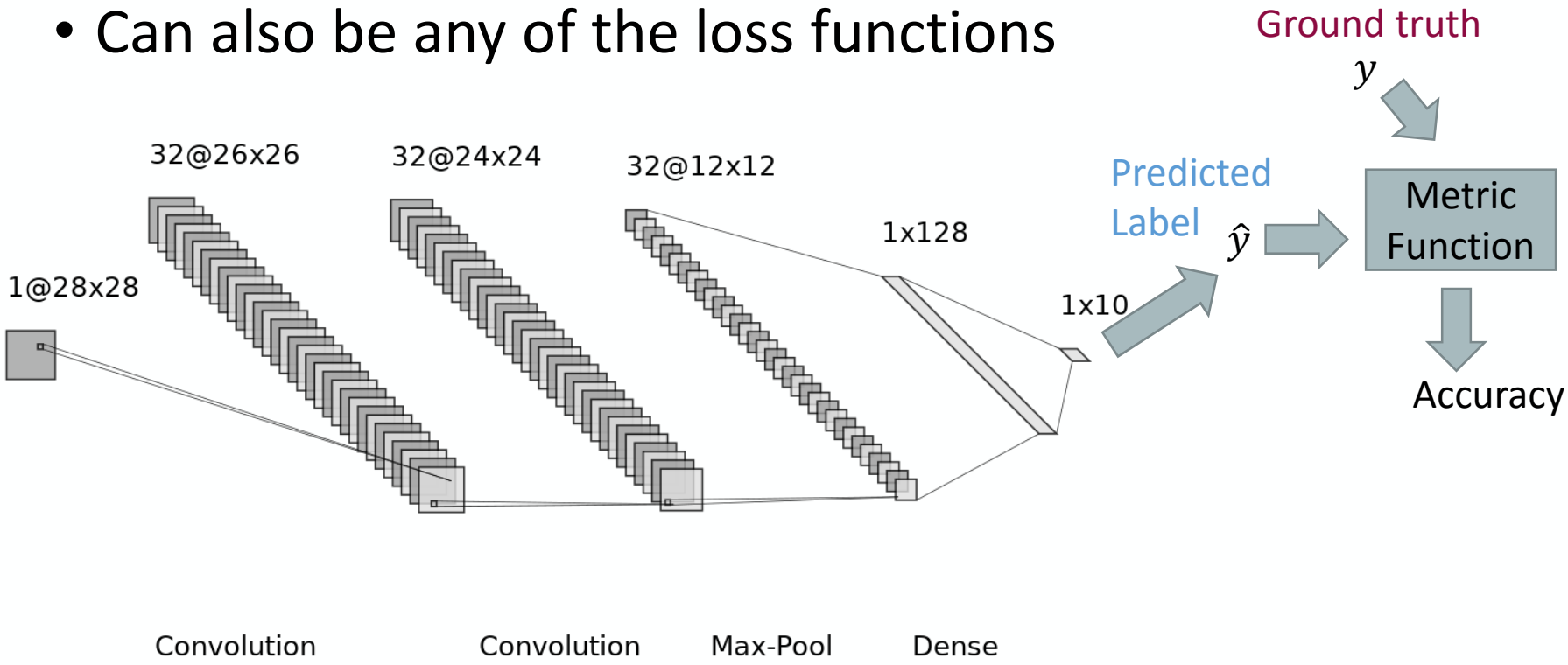
Optimizers

- Optimizers control adaptation of the parameters
  - Given the error, you know which direction to adapt the learned weights in backpropagation
  - How exactly you adapt those weights and how "fast" you adapt them is the job of the optimizer
  - The "Adam" optimizer is pretty good general-purpose optimizer, `optimizer='adam'`



Metrics

- Metrics are the metrics WE are interested in
- Generally are some form of accuracy
  `metrics=['accuracy']`
- Can also be any of the loss functions



Batches and Epochs

- We need to specify how long we are willing to wait for our network to train
- We don't expect to reach a loss of 0 (implying an accuracy of 100%)
- May other simulations invoke the notion of an "iteration"
- We cannot necessarily operate on our entire training set at once (due to memory constraints)
  - Batch: break up the training data into smaller, more manageable chunks (this is where the "stochastic" in stochastic gradient descent comes in), we specify the size of the batches with the `batch_size` parameter
  - Epoch: however many batches are needed to visit every training sample once, i.e., an epoch processes $\left\lceil \frac{N_{samples}}{batch\_size} \right\rceil$ batches
- We specify the maximum number of epochs with the `epochs` parameter

But wait…
There's more!!!!

There's more!?!?!

- There are **many** other options available:
  - Other layer types
  - Other activation functions
  - Other loss functions
  - Other optimizers
  - Other options in all of the above
- Check out the keras documentation at https://keras.io for more details.