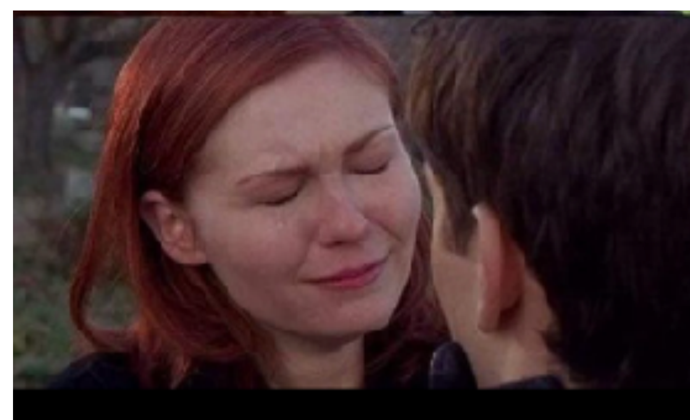
Basics of Convolutional Neural Network Visualization

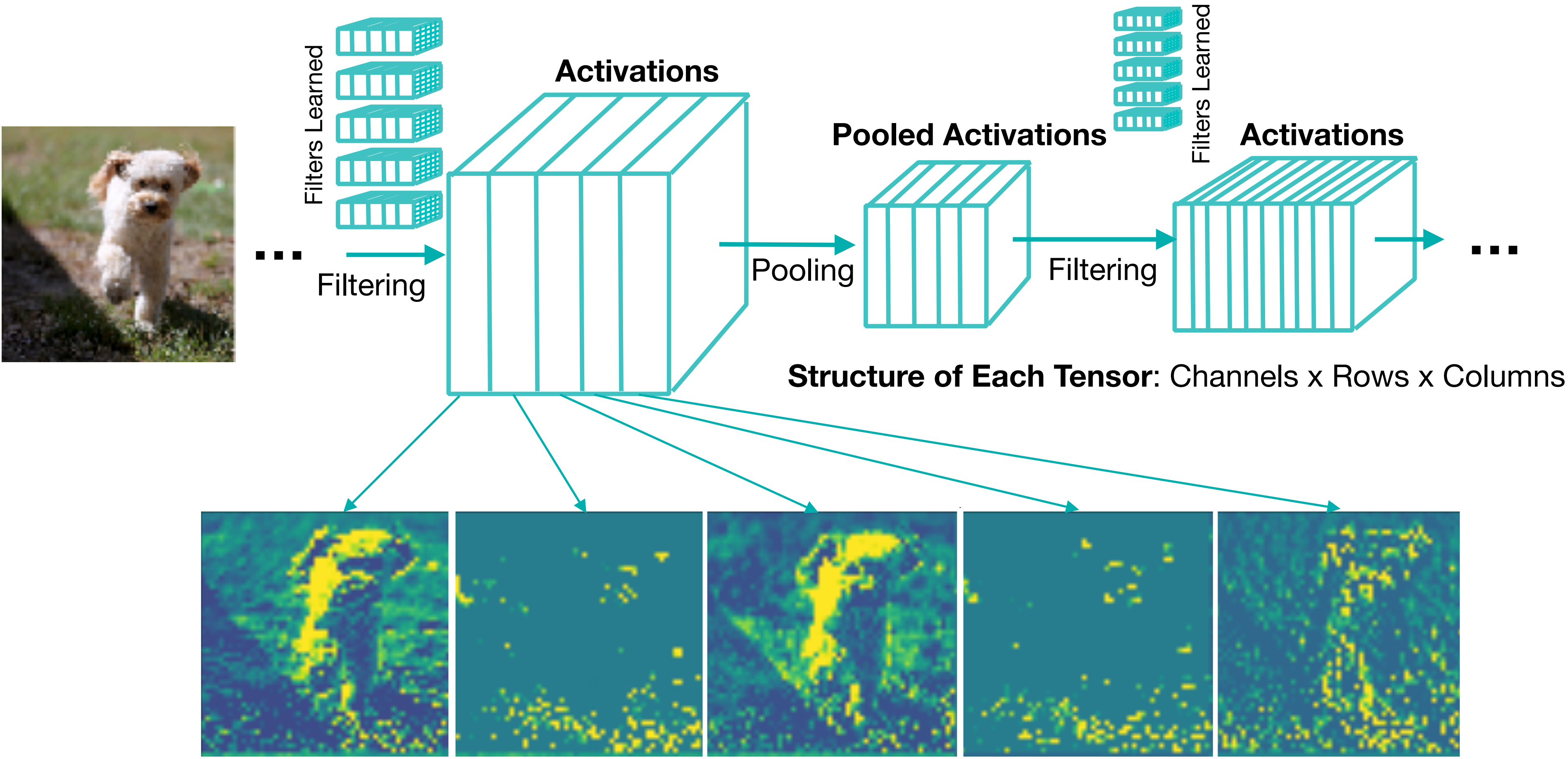


Tools to Visualize Neurons and Filters

- Visualize **Filter Activation**
  - What parts of the inputs activate each filter?
- Visualize **Filters**
  - What does each filter look like? Is it similar to other filters?
  - Can we excite a certain filter by updating the input image?
- **Heatmaps** of Class Activation
  - What part of an input image most influences each final output?

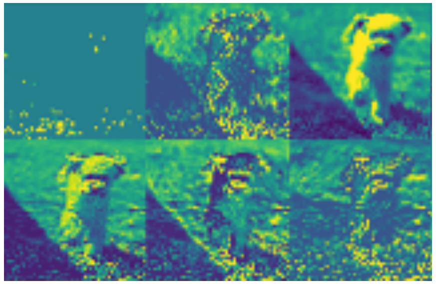
Visualizing Intermediate Activations

- Look layer by layer
- **Assume**: each filter learns something useful



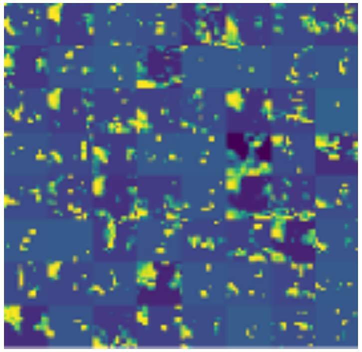
Visualizing Intermediate Activations

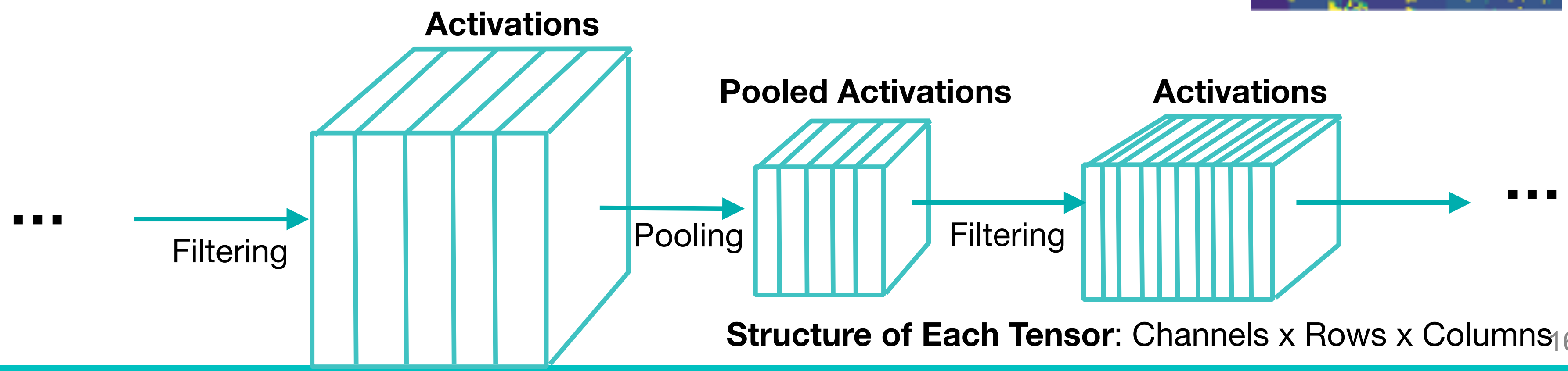
- **Recall**: general structure of most CNNs
  - Small kernels throughout (3x3)
  - Filtering followed by Pooling (spatial downsampling)
  - More filters in later layers



Early Activations are larger but not as numerous

Later Activations are smaller and more numerous

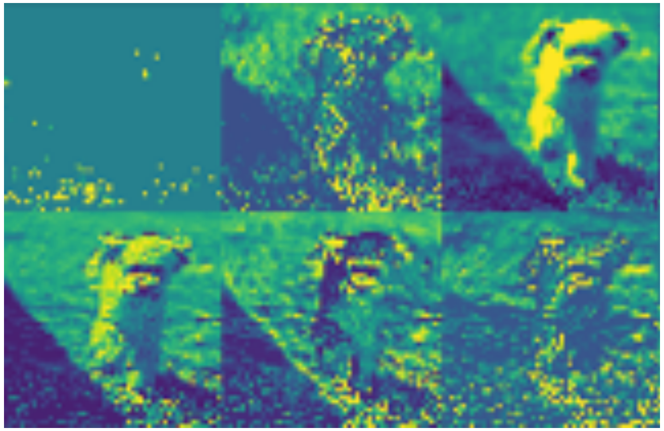




Structure of Each Tensor: Channels x Rows x Columns

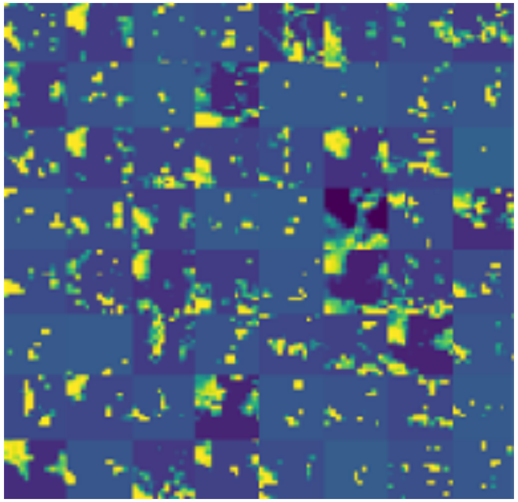
Visualizing Intermediate Activations

- **Result:** Information Distillation Pipeline
  - Deeper layers have more abstract triggers
  - Deeper activations are increasingly sparse
  - Early layers are texture and edge detectors
  - Notion of "High Level Abstraction," has biological motivation



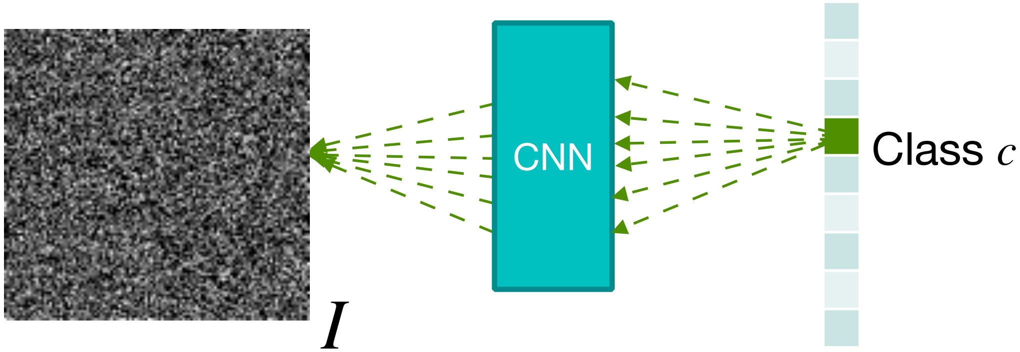
Early Activations are larger but not as numerous

Later Activations are smaller and more numerous

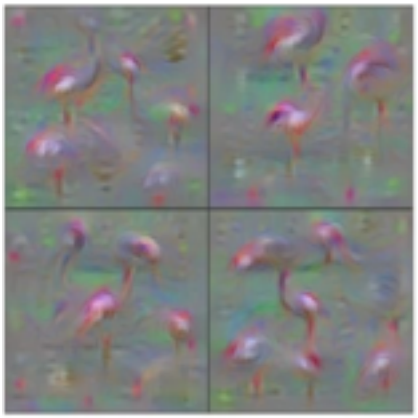


Visualizing Filters: Class Neuron

- **Idea:** What Maximally Activates a Class Output?
  - Gradient Ascent in the Input Space



$$I \leftarrow I + \eta \nabla f_c(I)$$


Flamingo

where c is a specific neuron in output layer

f is the neural network function

I is the input image, init to zeros (or random)

∇ is the gradient of f_c w.r.t I

CNN weights stay unchanged

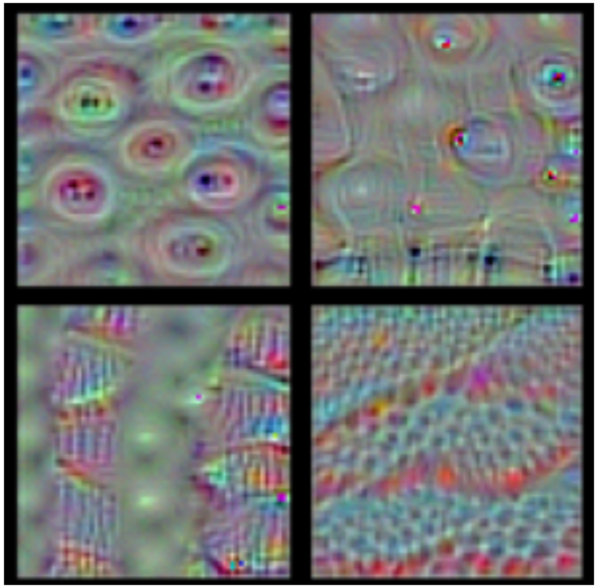
http://cs231n.github.io/understanding-cnn/

Visualizing Filters: Maximal Activations

- **Idea:** What Maximally Activates a **Filter**?
  - **Again**: Gradient Ascent in the Input Space

$$I \leftarrow I + \eta \sum_{i,j} \nabla f_n(I)_{i,j}$$

"trick" use norm of gradient



where n is a specific **filter** in a layer

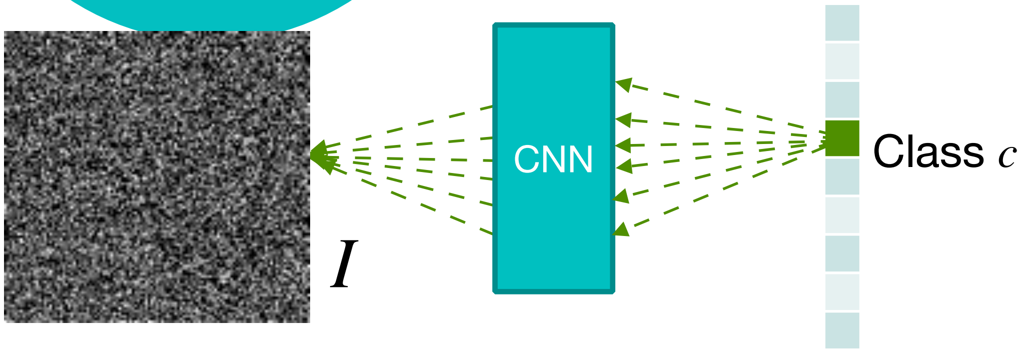
f is the function to n^{th} filter in layer

Visualizing ConvNets

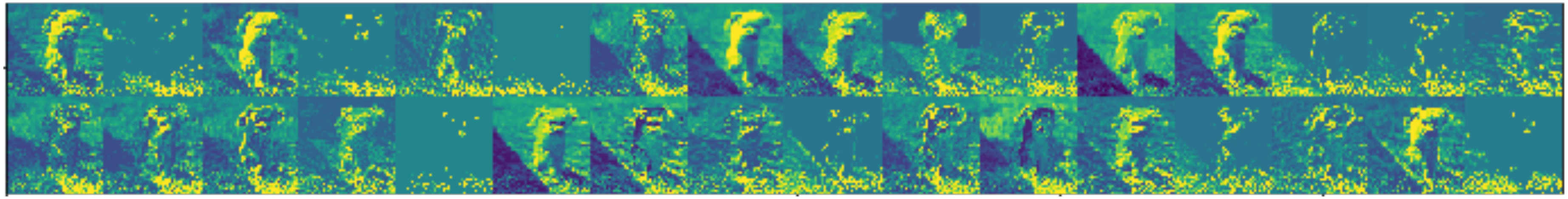
Part One: Filter Activations

Part Two: Image Gradients

Google AI



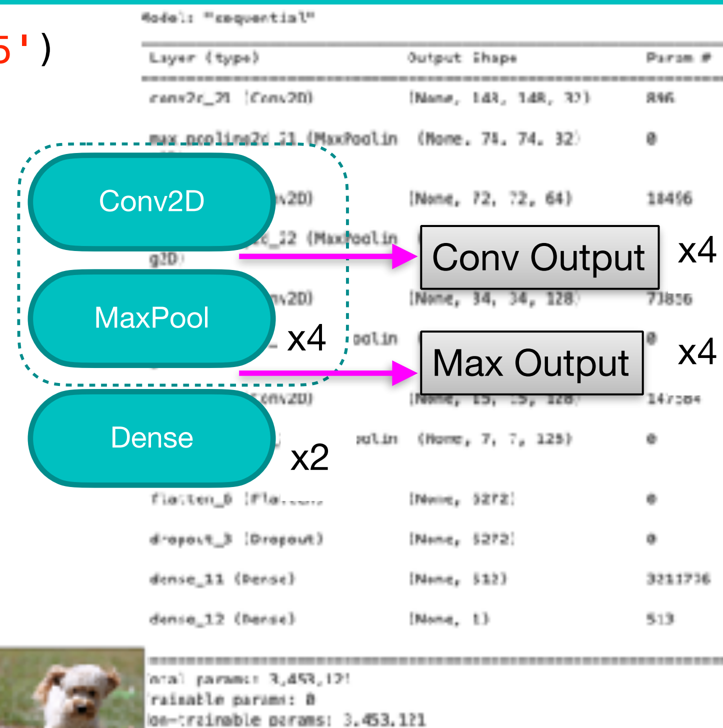
$$I \leftarrow I + \eta \nabla f_c(I)$$



Code Available: 04 LectureVisualizingConvnets.ipynb
activation-demo

Guided Demonstration: Visualize Channels

```
model = load_model('models/cats_and_dogs_small_2.h5')
for layer in model.layers:
    layer.trainable = False


def load_image_as_array(url, size=(150, 150)):
    … load and resize image …

def prepare_image_for_display(img, norm_type='max'):
    … normalize image, convert to numpy …
    return new_img.astype('uint8')


img_tensor = load_image_as_array(img_url)
img_tensor = np.expand_dims(img_tensor, axis=0)
plt.imshow(prepare_image_for_display(img_tensor))
```



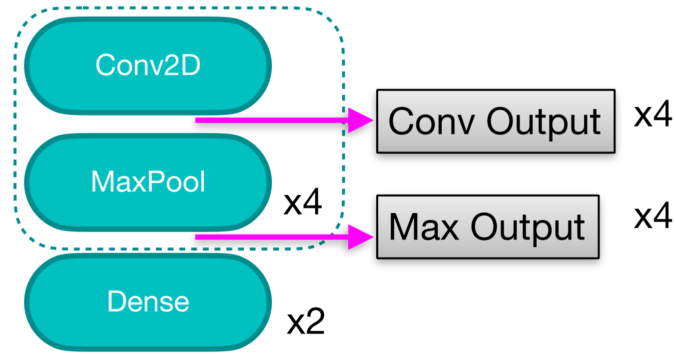
```
# Extract top 8 layers:
lay_outputs = [layer.output for layer in model.layers[:8]]
activation_model = models.Model(inputs=model.input,
                                outputs=lay_outputs)
activations = activation_model.predict(img_tensor)
[print(x.shape) for x in activations]
```

Create Model with many Outputs

```
(1, 148, 148, 32) conv2d
(1, 74, 74, 32)   maxpool
(1, 72, 72, 64)   conv2d
(1, 36, 36, 64)   maxpool
(1, 34, 34, 128)  conv2d
(1, 17, 17, 128)  maxpool
(1, 15, 15, 128)  conv2d
(1,  7,  7, 128)  maxpool
```

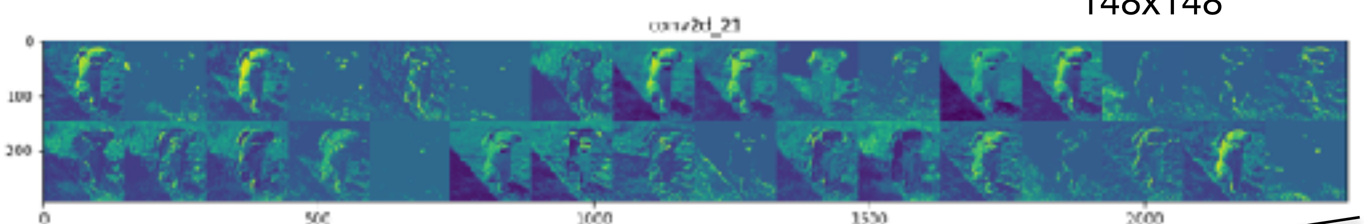
Guided Demonstration: Visualize Channels

```
activations = activation_model.predict(img_tensor)

for layer_activation in activations:
    … get activation shapes …

    display_grid = np.zeros(…preallocate grid…)
   # fill in grid with activations
   for each row and col of the grid:
       channel_image = layer_activation[…channel…]
       channel_image = …normalize for display…
       display_grid[…] = channel_image

   plt.imshow(display_grid, … options …)
```

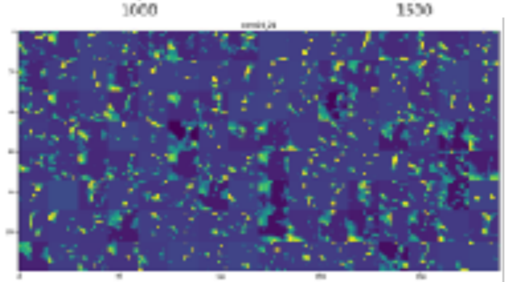


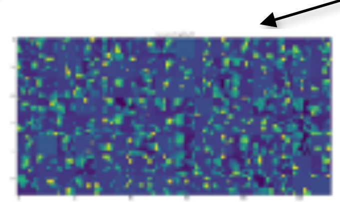

```
(1, 148, 148, 32) conv2d
(1, 74, 74, 32)   maxpool
(1, 72, 72, 64)   conv2d
(1, 36, 36, 64)   maxpool
(1, 34, 34, 128)  conv2d
(1, 17, 17, 128)  maxpool
(1, 15, 15, 128)  conv2d
(1,  7,  7, 128)  maxpool
```

32 images, 148x148



128 images, 15x15



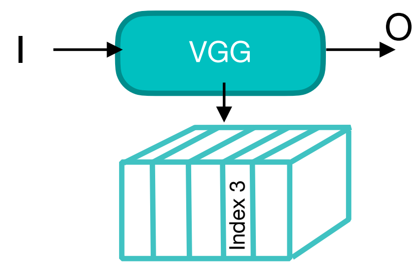


128 images, 7x7

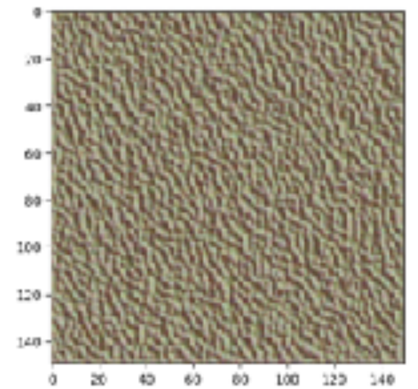
Optimize Filters for Input

```
model = VGG16(weights='imagenet', include_top=False, input_tensor=None)
[layer.trainable = False for layer in model.layers]

… Select a layer and channel to visualize …
layer_name = 'block1_conv1', filter_index = 3
layer_output = model.get_layer(layer_name).output
new_model = Model(inputs=model.input, outputs=layer_output)


# trainable tensor variable
I = tf.Variable(np.zeros(…), … options …)

for i in range(EPOCHS):
    with tf.GradientTape(watch_accessed_variables=False) as tape:
        tape.watch(I)
        channel_out = new_model(I)[:,:,:, filter_index]
        filter_output_to_maximize = tf.reduce_mean(channel_out)

    grad_fn = tape.gradient(filter_output_to_maximize, I)
    grad_fn /= normalize for better stability
    I += grad_fn # one iteration of maximizing
```

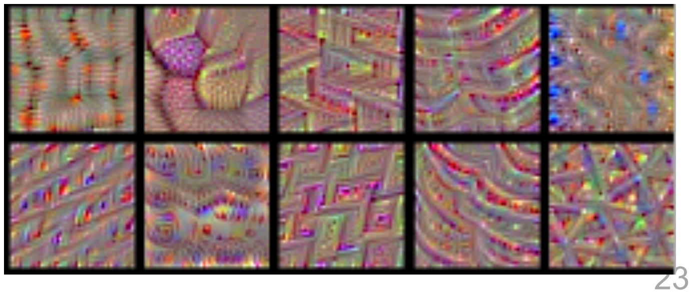
$$I \leftarrow I + \frac{\eta}{|f_n(I)|} \sum_{i,j} \nabla f_n(I)_{i,j}$$



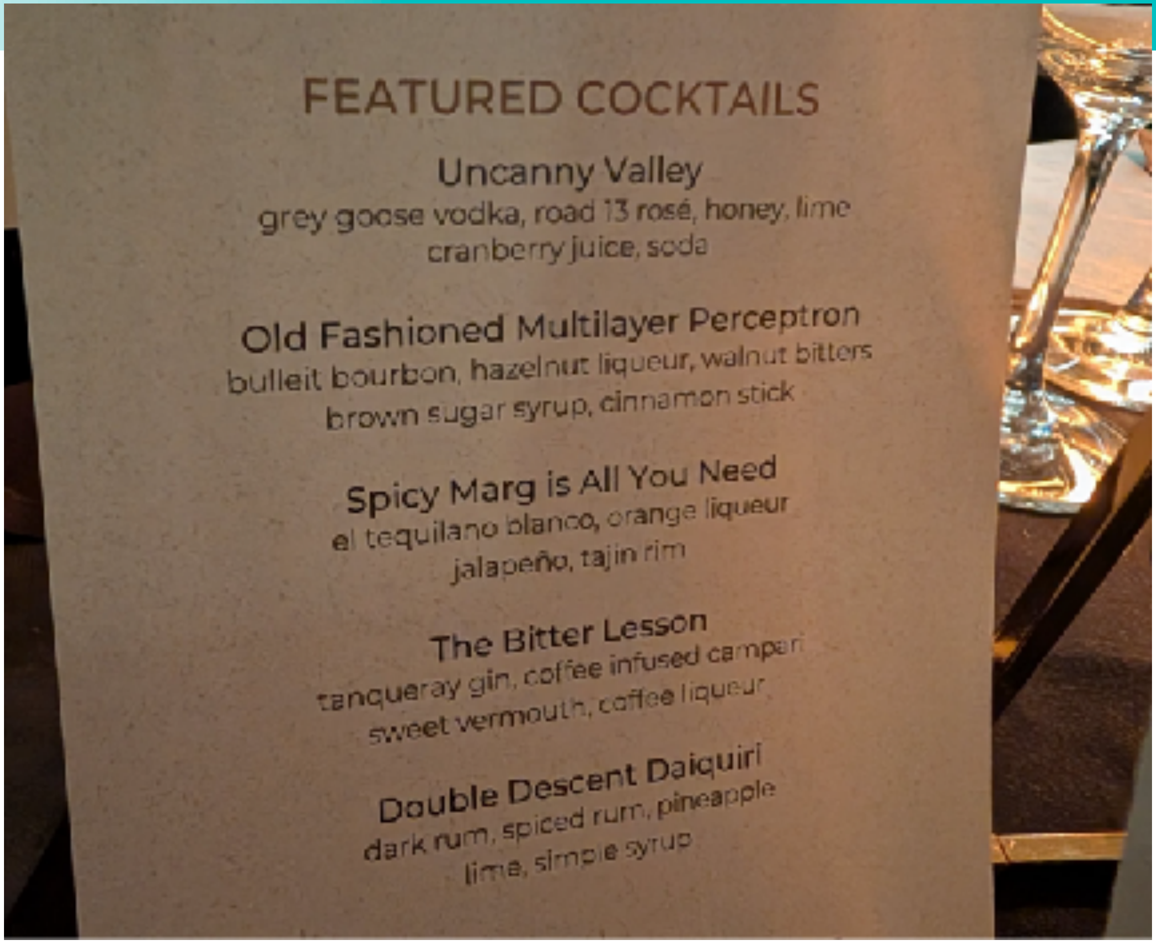
Block 1, Conv1



Block 4 Conv 1, Indices 0-9



Gradient Class Activation Mapping



Class Activation Mapping (CAM)

- **Idea:** What areas of the image contributed most to the classification result?
- Also, for each class, what areas of the image exhibit features of that class?
- Use change in output, w.r.t. final conv layer

normalize by h x w of A

final layer output in response to image I
c is class of interest

$$\alpha_k^c = \frac{1}{|A_k^{(L)}|} \sum_{i,j} \frac{\partial f_c(I)}{\partial A_{i,j,k}^{(L)}}$$

final convolutional layer, L, activations
for row, column, channel

gradient weight for channel k and class c in layer L
k in $1 \ldots K$ activations in final layer

Ramprasath et al., **Grad-CAM:** Visual Explanations from Deep Networks via Gradient-based Localization

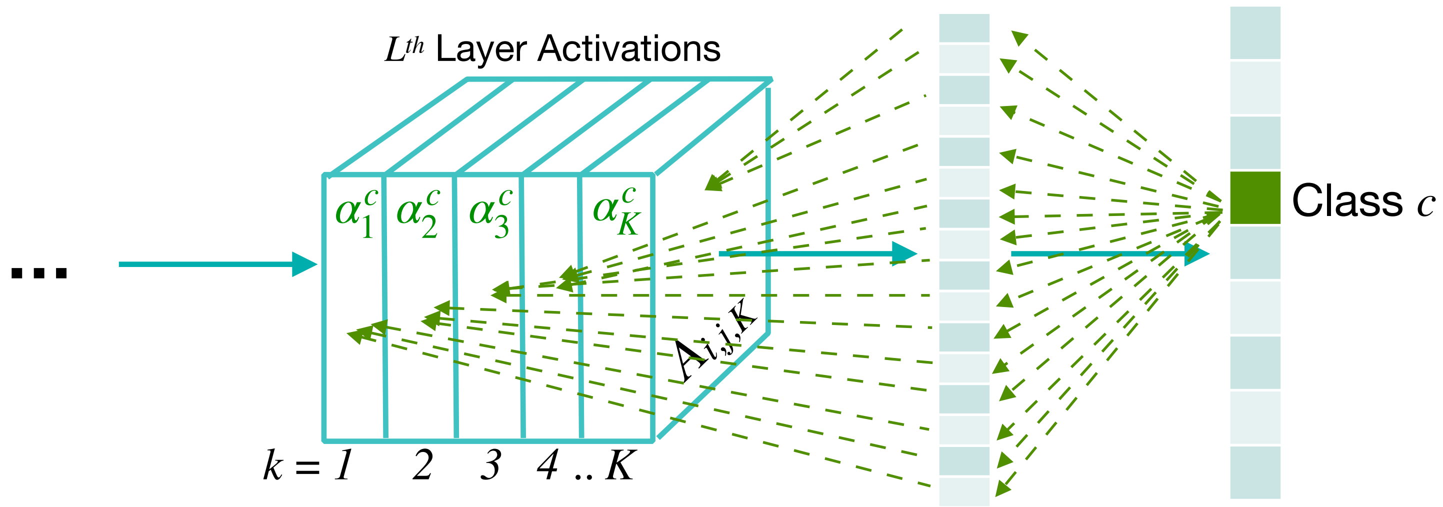
Class Activation Mapping (CAM)

$$\alpha_k^c = \frac{1}{|A_k^{(L)}|} \sum_{i,j} \frac{\partial f_c(I)}{\partial A_{i,j,k}^{(L)}}$$

final layer output in response to image I
c is class of interest

final convolutional layer, L, activations
for row, column, channel

gradient weight for channel k and class c in layer L
k in $1 \ldots K$ activations in final layer



Sensitivity of Class to Activations

Ramprasath et al., **Grad-CAM:** Visual Explanations from Deep Networks via Gradient-based Localization

Class Activation Mapping (CAM)

$$\alpha_k^c = \frac{1}{|I \times J|} \sum_{i,j} \frac{\partial f_c(I)}{\partial A_{i,j,k}^{(L)}}$$

final layer output in response to image I
c is class of interest

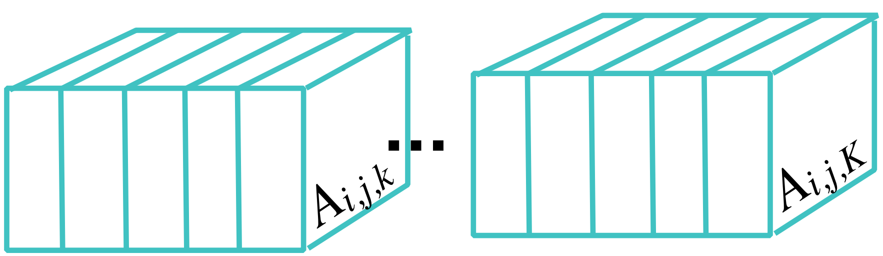
final convolutional layer, L, activations
for row, column, channel

gradient weight for channel k and class c in layer L
k in $1 \ldots K$ activations in final layer

Heatmap, S, is the **weighted sum** of final layer activations:

$$S_{i,j} = \frac{1}{S_{max}} \sum_{k} \phi(\alpha_k^c A_{i,j,k}^{(L)})$$

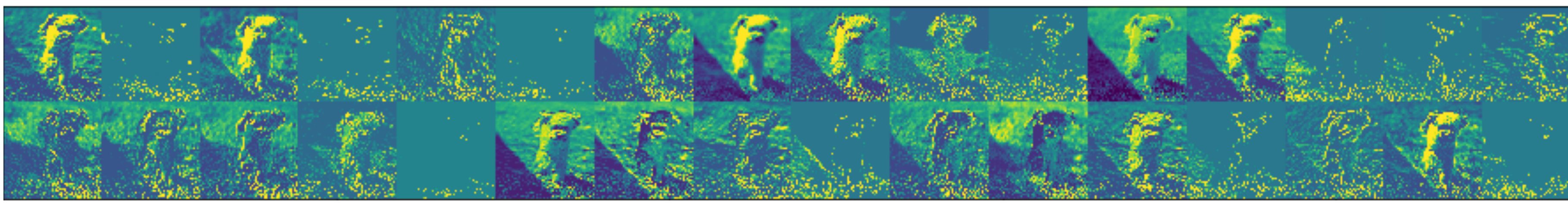
relu activation

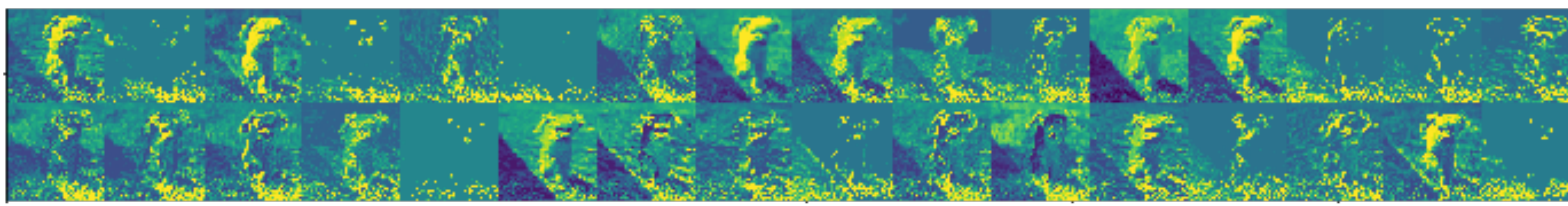


Ramprasath et al., **Grad-CAM:** Visual Explanations from Deep Networks via Gradient-based Localization

Visualizing ConvNets

Part Three: Grad-CAM





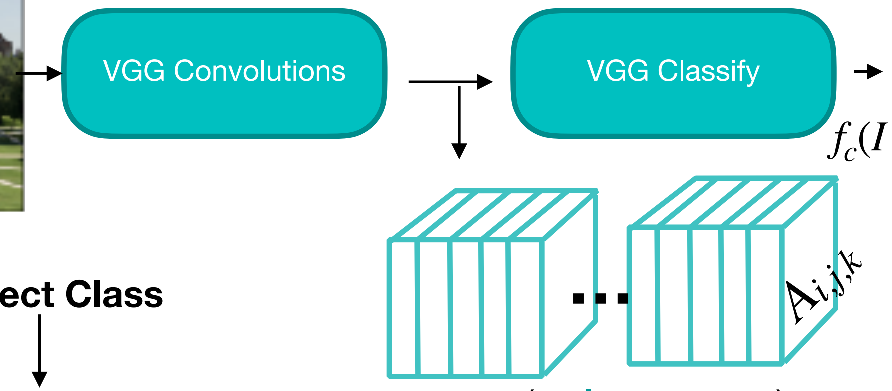
```
Code Available: 04 LectureVisualizingConvnets.ipynb
activation-demo
```

Grad CAM, Guided Demo

$$\alpha_k^c = \frac{1}{|I \times J|} \sum_{i,j} \frac{\partial f_c(I)}{\partial A^{(L)}_{i,j,k}}$$

$$S_{i,j} = \frac{1}{S_{max}} \sum_{k} \phi(\alpha_k^c \cdot A^{(L)}_{i,j,k})$$

```
preds = model.predict(img)
decode_predictions(preds)
```

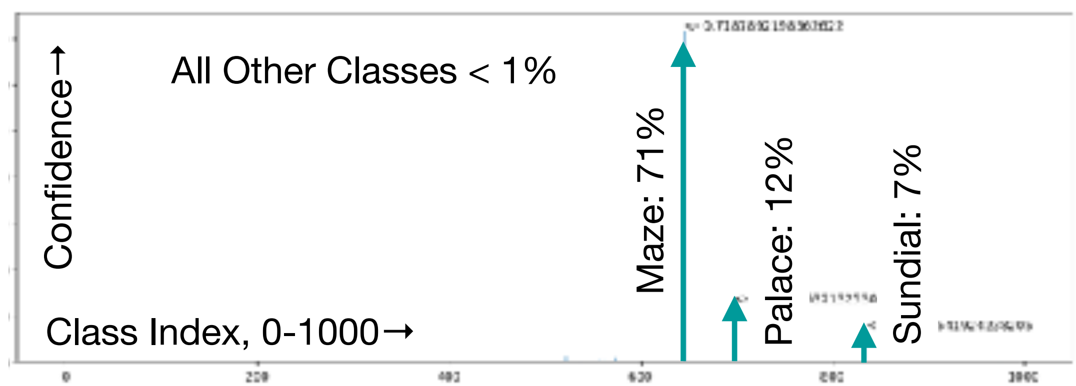



```
Aijk_kt = model.(…final conv…).output
new_mod = Model(inputs=model.input,
          outputs=[Aijk_kt, model.output])
```

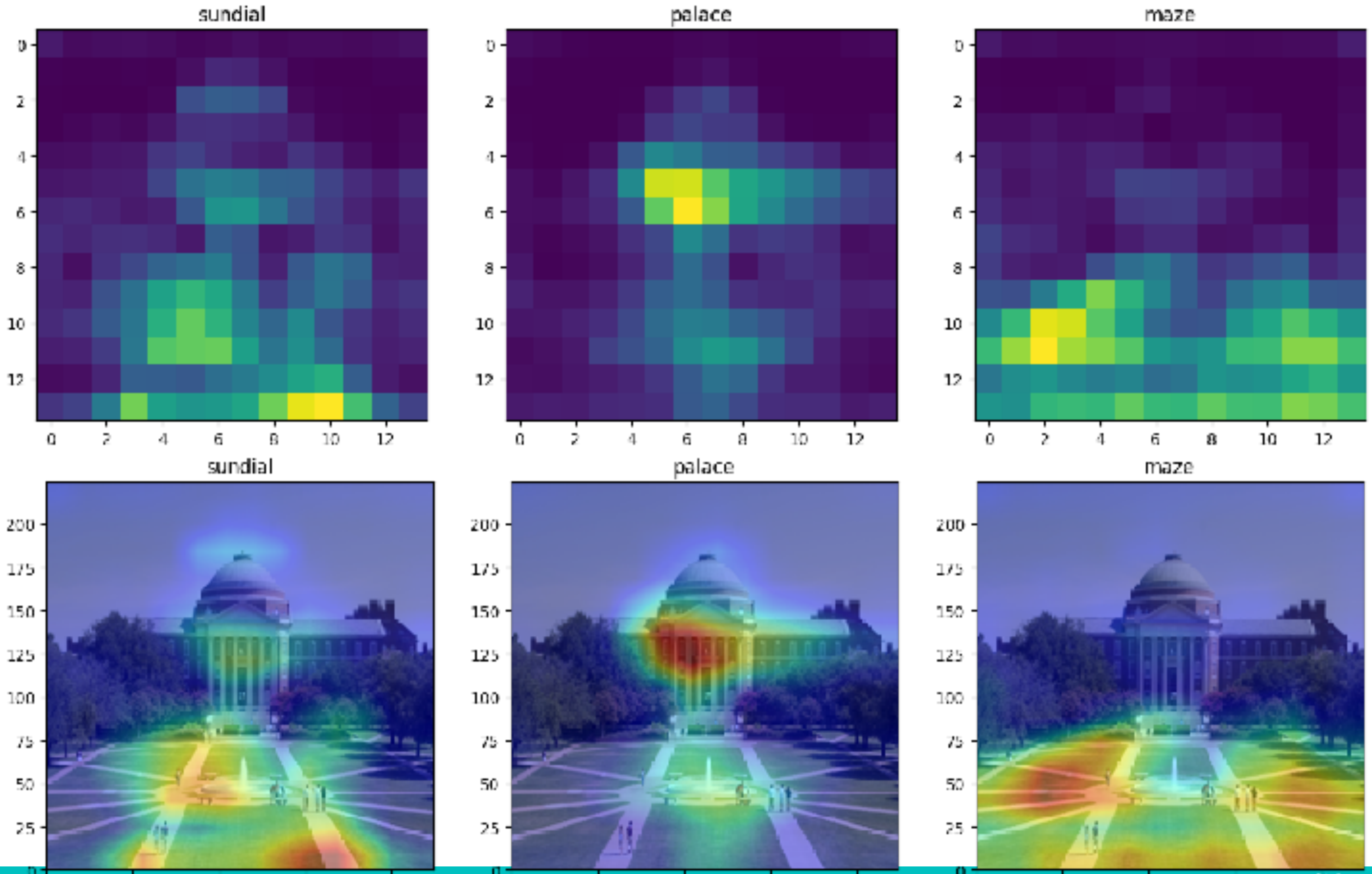
Select Class

```
with tf.GradientTape() as tape:
    Aijk_tf, fc_tf = new_mod(I)
    sum_fc = tf.reduce_mean(fc_tf[:, class_idx])

grad_var = tape.gradient(sum_fc, Aijk_tf)
alpha_k = tf.reduce_mean(grad_var,
                         axis=(0, 1, 2)) (Batch x H x W)
```

```
alpha_k = alpha_k.numpy()
A = Aijk_tf.numpy()
for chan in N_channels:
    A[:,:,:,chan] *= alpha_k[chan]
    A[:,:,:,chan] = …ReLU of A…

S = np.sum(A, axis=-1)
S /= np.max(S)
```



Grad CAM, Visualized



Lecture Notes for

Neural Networks and Machine Learning

CNN Visualization

Next Time:
CNN Circuits

Reading: OpenAI Circuits