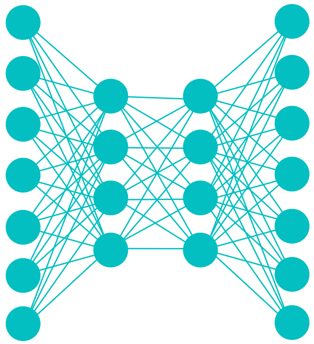
Lecture Notes for

# Neural Networks and Machine Learning

## CNN Visualization

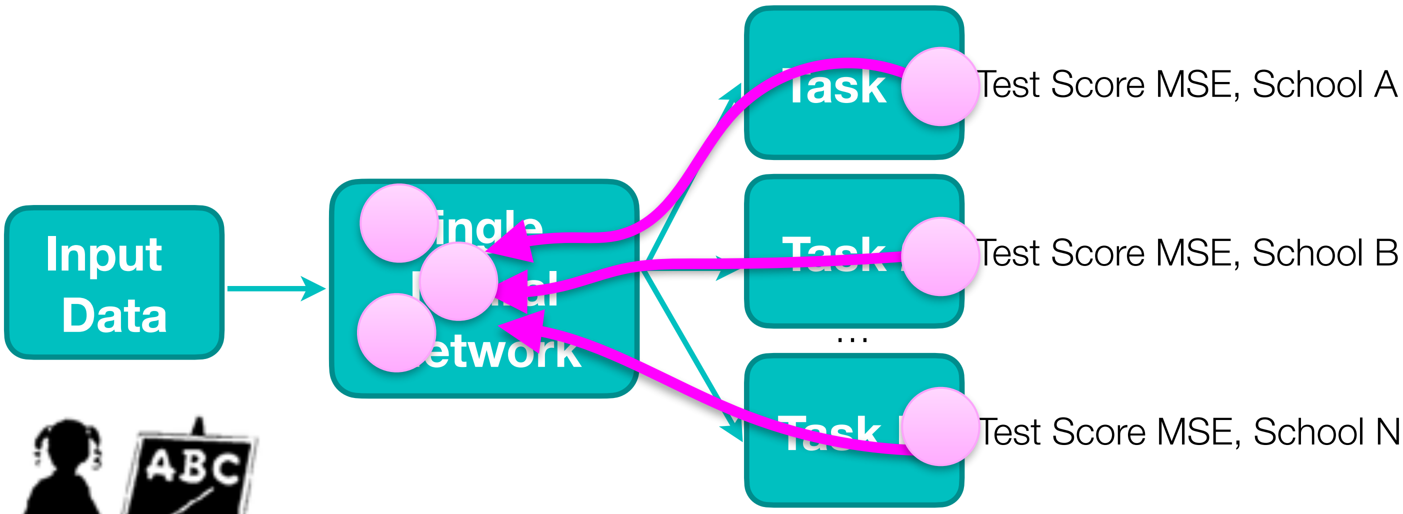


# Logistics and Agenda

- Logistics
  - None
- Agenda
  - Multi-Task Demo, Revisit
  - Visualizing Convolutional Architectures and Demo
- Next Week:
  - Circuits in CNNs

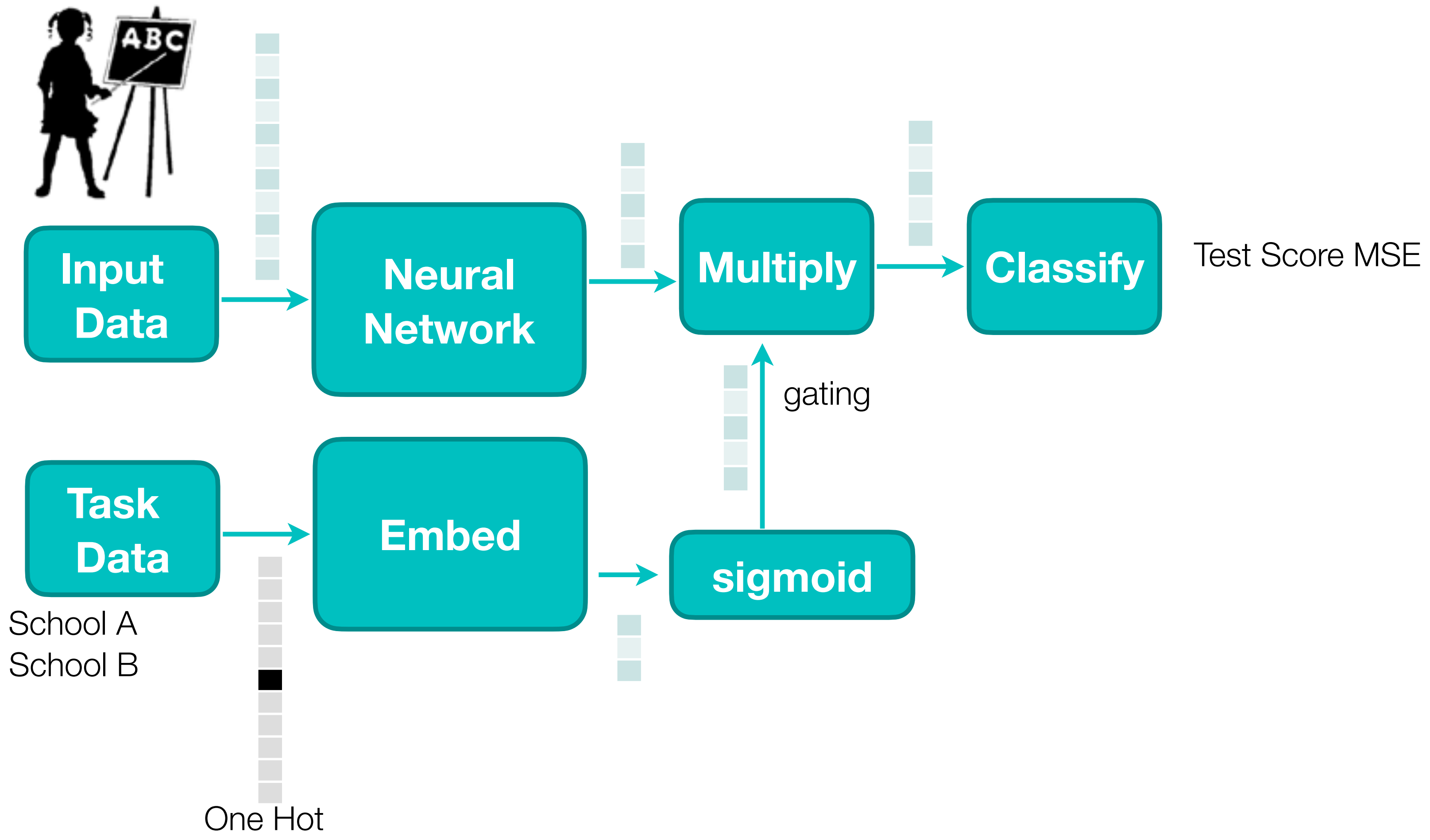
# Multi-task Optimization, Review

## Single Task Label per Input




Accumulate Gradients, Batch updates across multiple tasks

# An alternative: Task-Gating, Review



# Multi-Task Learning

## School Data, Computer Surveys

Traian-Pop Traian Pop

Sponsor Follow

LukeWood Luke Wood

KerasCV Author, Full Time Keras team member & Machine Learning researcher @ Google, Part Time UCSD Ph.D student

PRO

**Testing out a new idea: structured demonstration with mix of code and pseudocode…**

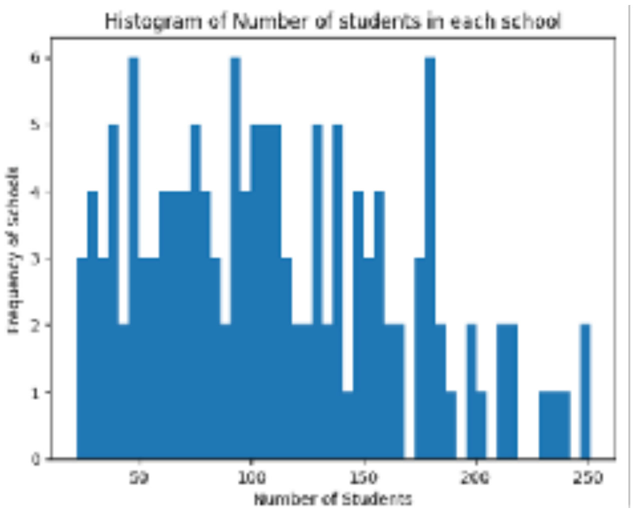
`LectureNotesMaster/03 LectureMultiTask.ipynb`

# Demonstration Overview

- Scale features and outputs
- Organize schools into dictionaries of Bunch data:
- Baseline: one model per school (limited training)
- Baseline: one model overall (nothing per school)
- Multi-task: shared model with school specific layer
- Gated: one model with task gating

```
feature_scaler = StandardScaler()
output_scaler = StandardScaler()

X = feature_scaler.fit_transform(X)
y = output_scaler.fit_transform(y)
```

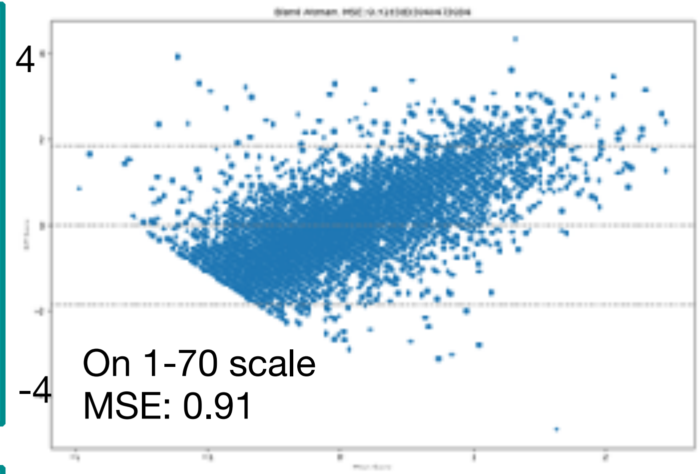
```
sid = "School {}".format(i + 1)
tasks[sid] = Bunch(data=X[start:end],
                   target=y[start:end],
                   DESCR=descr)
```



# Baseline Modeling

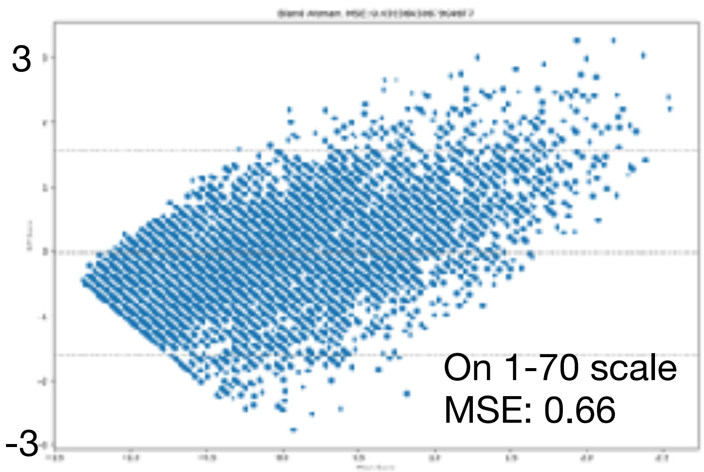
```
for sid in tasks.keys():

    mlp = Sequential()
    mlp.add( … layers …)
    mlp.compile( …loss and optimizer… )
    mlp.fit(X_train[sid], y_train[sid], … )

    # save the output results
    yhat_mlp[sid] = mlp.predict(X_test[sid])
```

Per school model




```
X_train_all, X_test_all = … gather all data …

mlp = Sequential()
mlp.add( … layers …)
mlp.compile( …loss and optimizer… )
mlp.fit(X_train_all, y_train_all, … )

# save the output results
yhat_mlp_all = mlp.predict(X_test_all)
```

Single Model

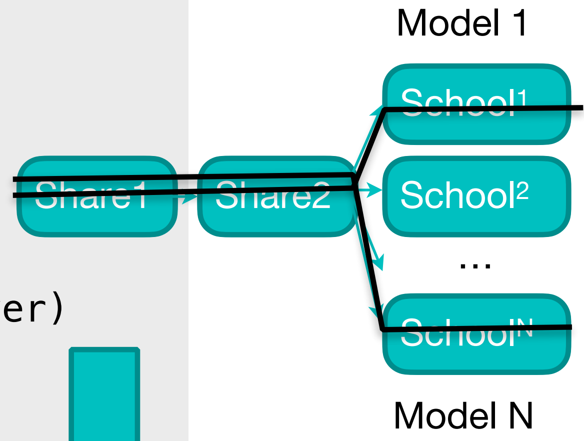


# Multi-task Modeling, setup

```
w1, w2, w_output = … get general model weights …

inputs = Input(… single input to CG …)
shared_input = Dense(… trainable=False)(inputs)
shared_mlp = Dense(…, trainable=False)(shared_input)
models = dict() # models for each task

for sid in tasks.keys():
    output_layer = Dense(1, …)(shared_mlp) # new layer

    models[sid] = Model(inputs=inputs, outputs=output_layer)

    shared1  = models[sid].get_layer('shared_input')
    shared2  = models[sid].get_layer('shared_middle')
    personal = models[sid].layers[-1]

    # set weights from the general model, as starting point
    shared1.set_weights(w1)
    shared2.set_weights(w2)
    personal.set_weights(w_output)

    personal.trainable = True
```

Shared Layers

Multi-task Layers

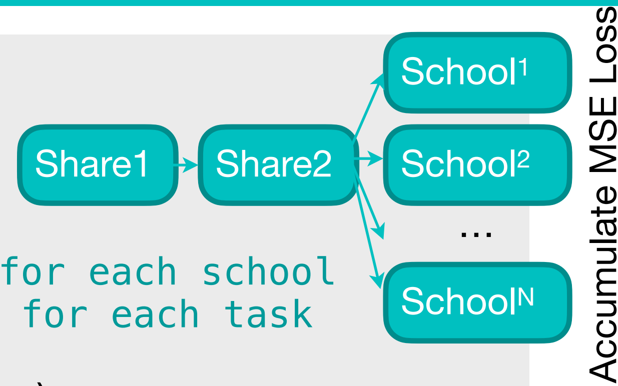


# Multi-task Modeling, setup

```
def step(keys, opt):
    # each key will be a separate school
    loss = {}, tapes = {}
    for sid in keys:
        with tf.GradientTape() as tape:
            # accumulate all the gradient updates for each school
            # make a prediction and calculate loss for each task
            tapes[sid] = tape # need to track
            preds      = models[sid]( X_train[sid] )
            loss[sid]  = mean_squared_error( y_train[sid], preds )
```

**Could weight the loss here to account for schools with low enrollment**

```
    # now batch update all the models with the gradients
    for sid in keys:
        grads = tapes[sid].gradient(loss[sid],
                                    models[sid].trainable_variables)
        opt[sid].apply_gradients(zip(grads,
                                    models[sid].trainable_variables))
```




Gradient Accumulation

```
opt = {} # separate optimizers per task
for sid in all_keys:
    opt[sid] = Adam()

for i in range(EPOCHS):
    shuffle(all_keys) # shuffle
    step(all_keys, opt) # optimize
```

Apply Fitting

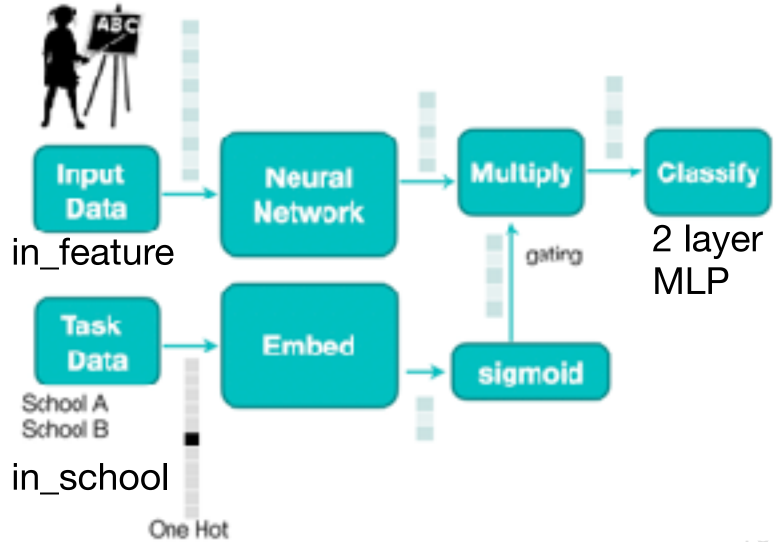
```
layer1, layer2 = … shared layers …
layer1.trainable = False
layer2.trainable = True

for i in range(EPOCHS):
    shuffle(all_keys) # shuffle
    step(all_keys, opt) # optimize
```

Fine tune

# Task Gating Model

```
in_feature = Input(… feature data …, name = 'student_features')
in_school = Input(… school id …, name = 'school_id')

num_schools = 139, embed_sz = 32
x_school = Embedding(input_dim=num_schools, output_dim=embed_sz, …)(in_school)
x_student= Dense(units=embed_size, … )(in_feature)

x = Multiply()([ sigmoid(x_school), x_student]) # gating

x = Dense( … )(x)
x_out = Dense(1, activation='linear', … )(x)

gated_mlp = Model(inputs=[in_feature, in_school], outputs=x_out)
```

```
gated_mlp.compile( … )
gated_mlp.fit({'student_features':X_train_all,
               'school_id':s_train_all },
                   y_train_all, … )

y_hat_gated = gated_mlp.predict(
                {'student_features':X_test_all,
                 'school_id':s_test_all })
```

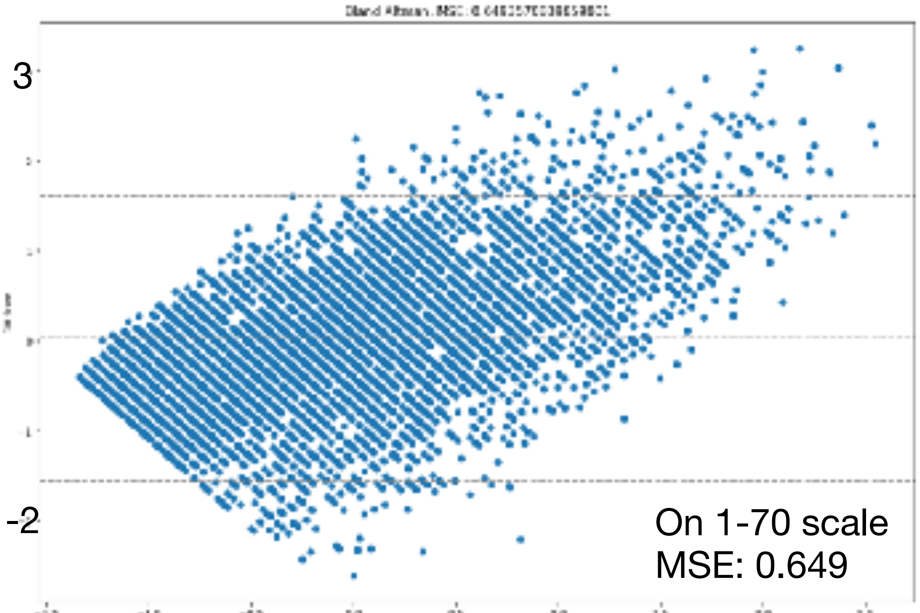


# Results (schools and survey data)

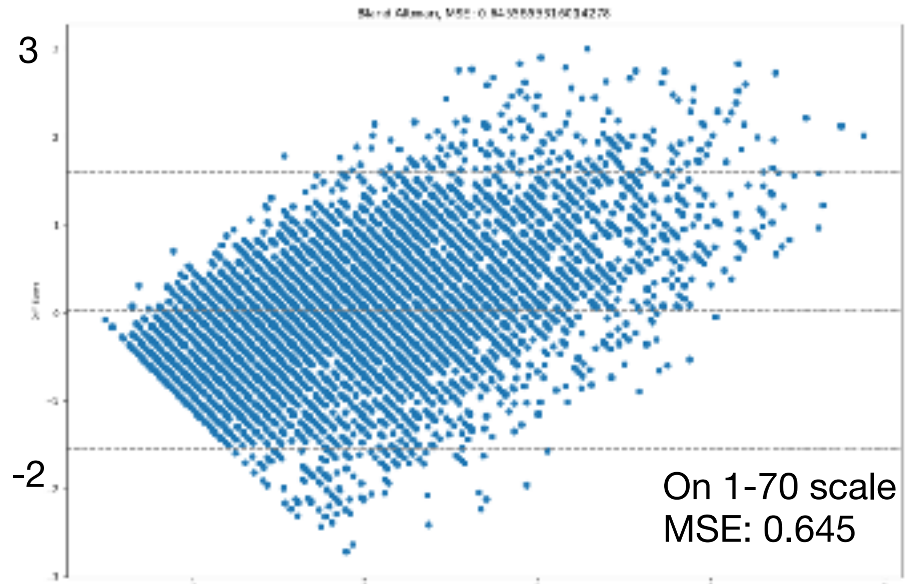
| School Model | MSE | Training Time |
|---|---|---|
| ***Per School*** | *0.916* | *3 min* |
| ***Single Model*** | *0.660* | *1.5 min* |
| ***Multi-task*** | *0.649* | *4 min* |
| ***Gated*** | ***0.645*** | ***1 min*** |

Not always as Clear Cut

| Survey | MSE | Train Time |
|---|---|---|
| ***Per School*** | *21.70* | *3 hours* |
| ***Single Model*** | *9.03* | *15 min* |
| ***Multi-task*** | *6.22* | *50 min* |




Multi-task, Fine tuned



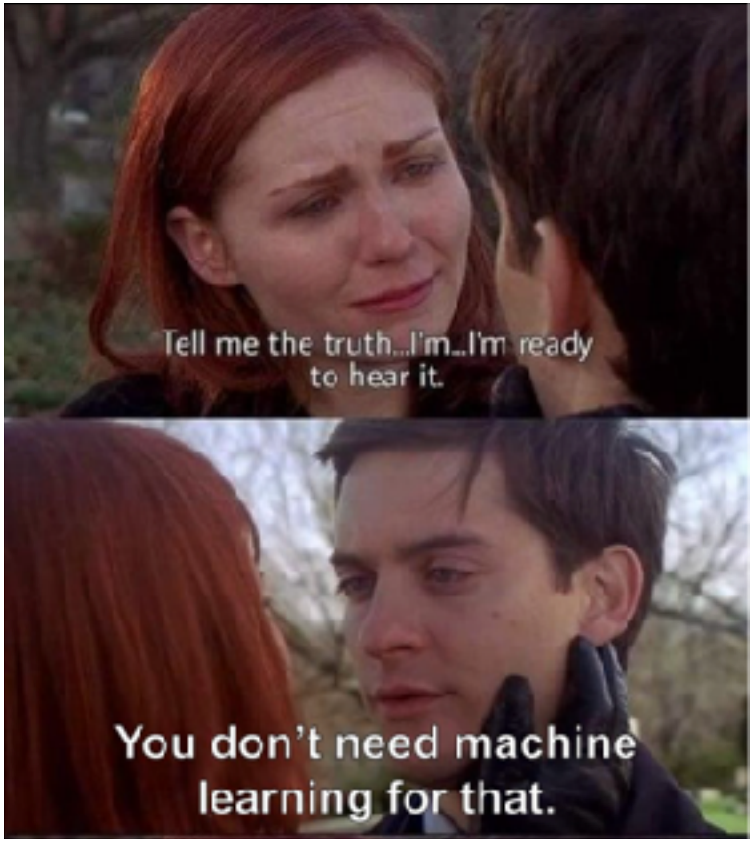

Gating Model

`LectureNotesMaster/03 LectureMultiTask.ipynb`

# Demonstration Comments?

- Pros/Cons

# Basics of Convolutional Neural Network Visualization



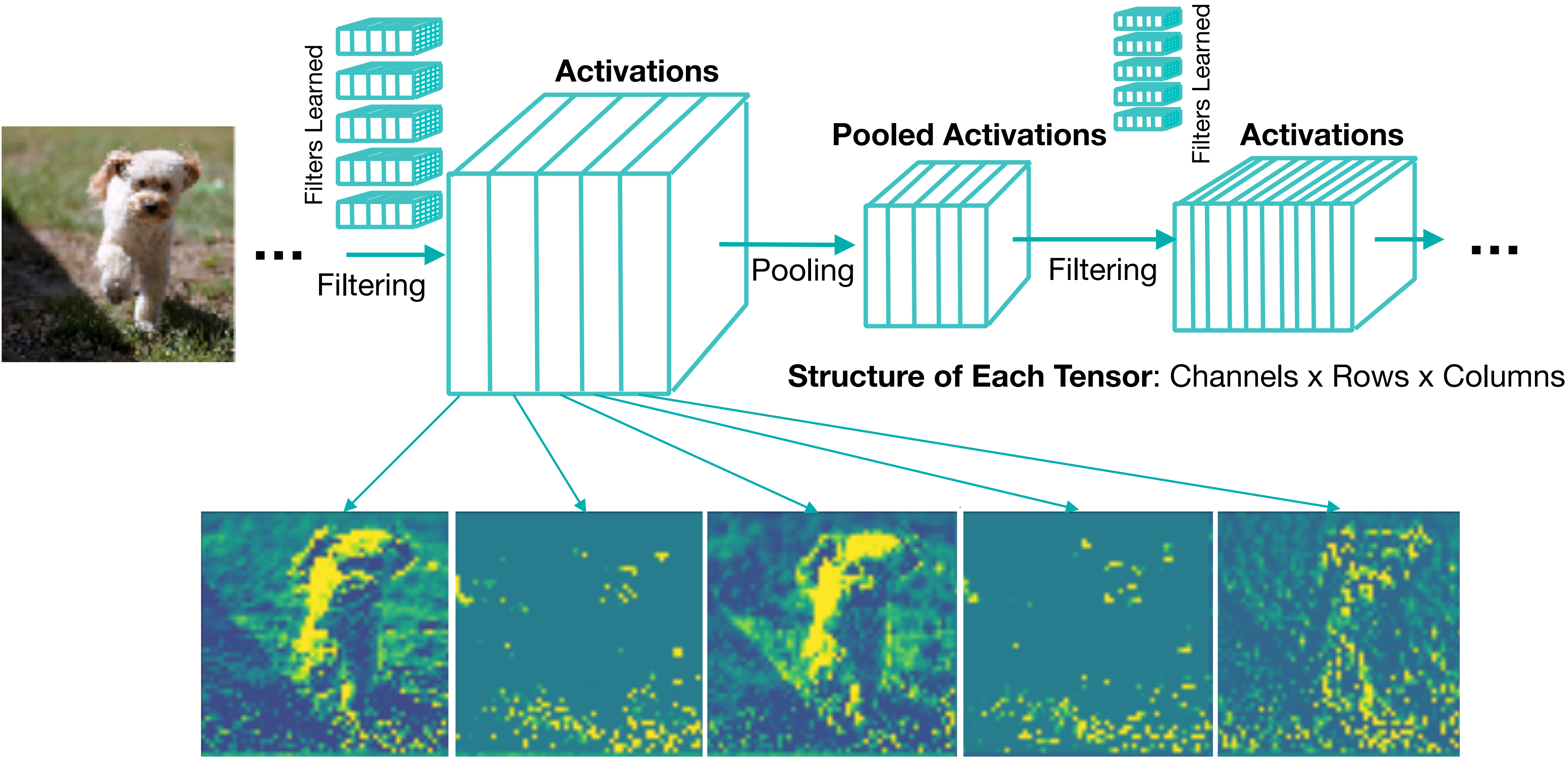

# Tools to Visualize Neurons and Filters

- 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



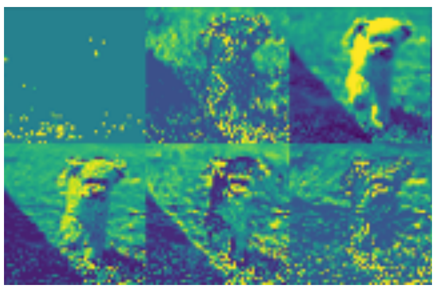
- Look layer by layer
- **Assume**: each filter learns something useful



# Visualizing Intermediate Activations

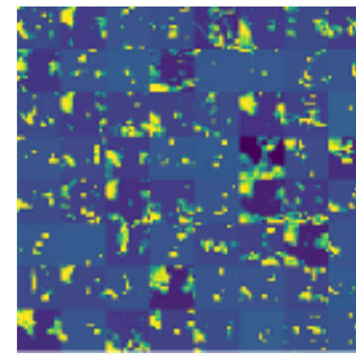


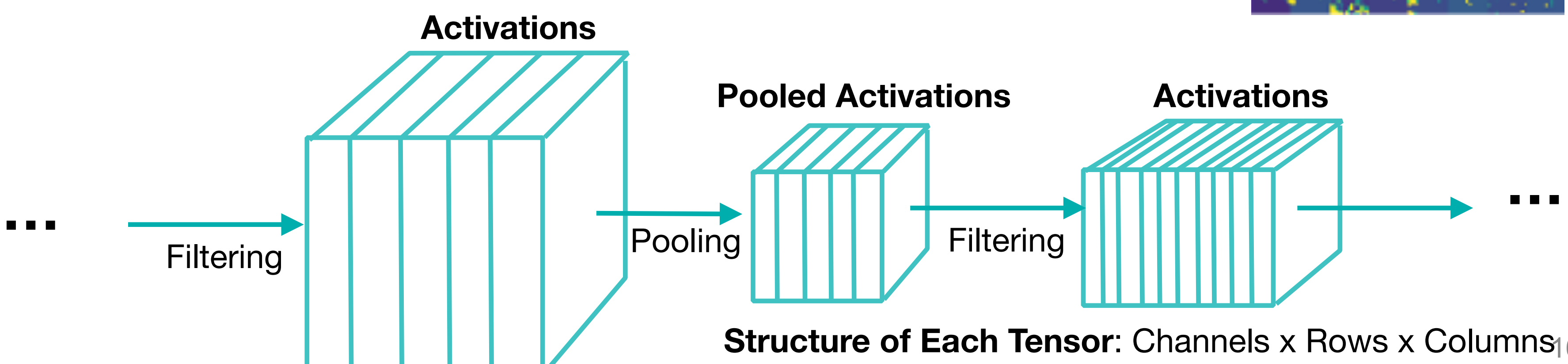
- **Recall**: general structure of most CNNs
  - Small kernels throughout (3x3)
  - Filtering followed by Pooling (spatial downsampling)
  - More filters in later layers



**Early Activations** are larger but not as numerous

**Later Activations** are smaller and more numerous



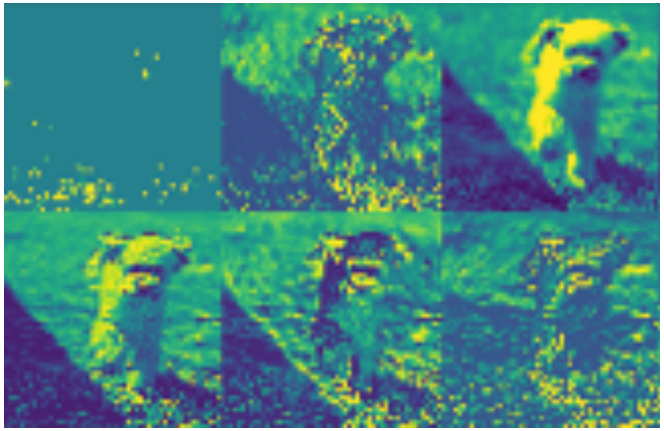



**Structure of Each Tensor**: Channels x Rows x Columns

# Visualizing Intermediate Activations

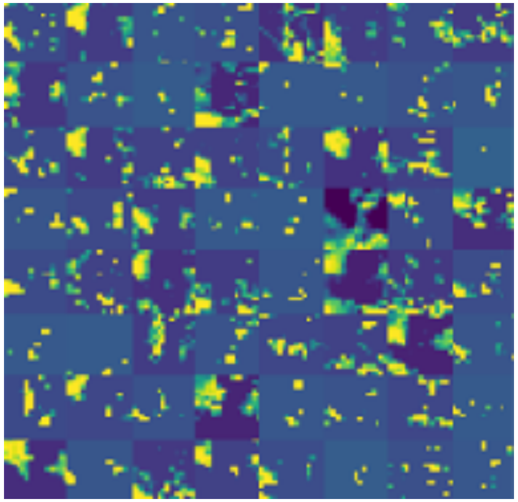


- **Result:** Information Distillation Pipeline
  - Deeper layers have more abstract triggers
  - Deeper activations are increasingly sparse
  - Early layers are texture and edge detectors
  - Notion of “High Level Abstraction,” has biological motivation



**Early Activations** are larger but not as numerous

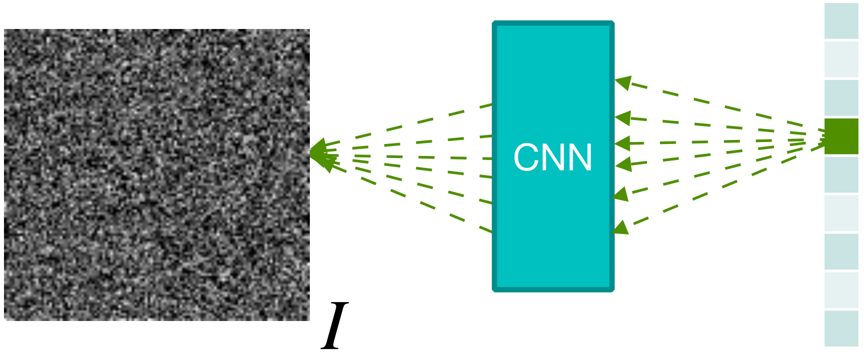
**Later Activations** are smaller and more numerous



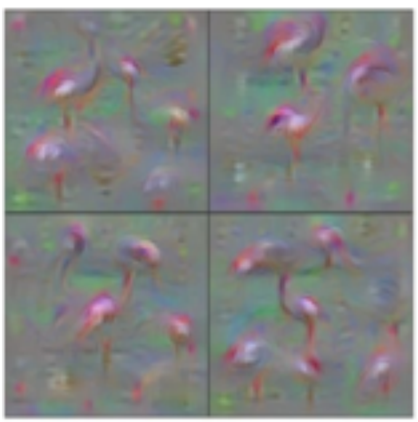
# Visualizing Filters: Class Neuron

Method Two

- **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)

$\nabla$ is the gradient of $f_c$ w.r.t $I$

CNN weights stay unchanged

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

# Visualizing Filters: Maximal Activations

Method Two

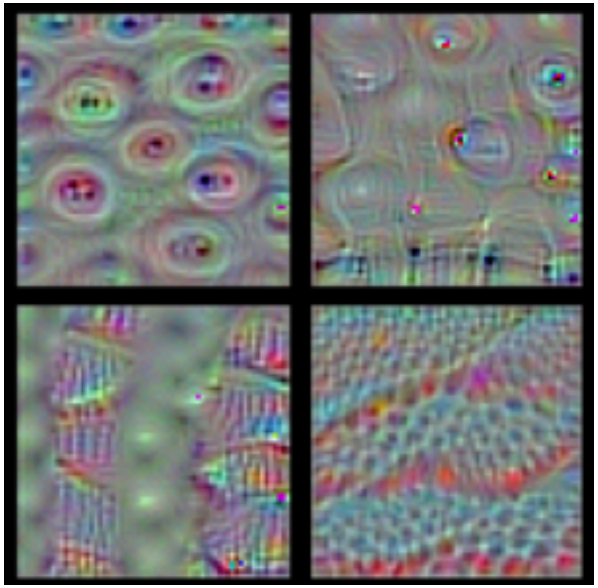
- **Idea:** What Maximally Activates a **Filter**?
  - **Again**: Gradient Ascent in the Input Space

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

**"trick"** use norm of gradient

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

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



19

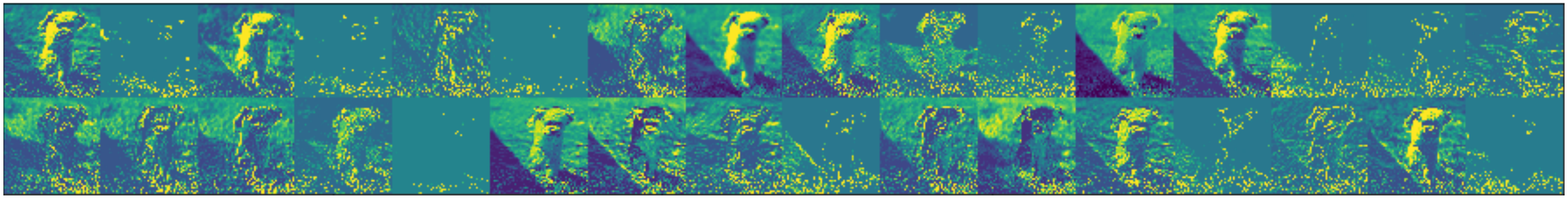
# Visualizing ConvNets

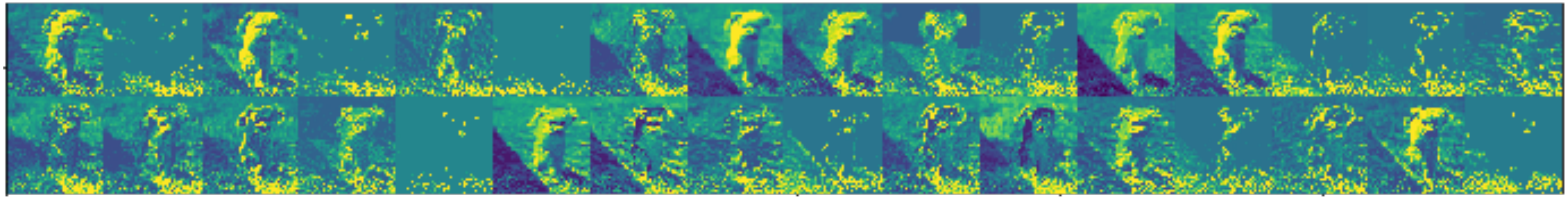
Part One: Filter Activations

Part Two: Image Gradients

Ian Stenbit

Google AI





```
Follow Along: 04 LectureVisualizingConvnets.ipynb
activation-demo
```

# 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

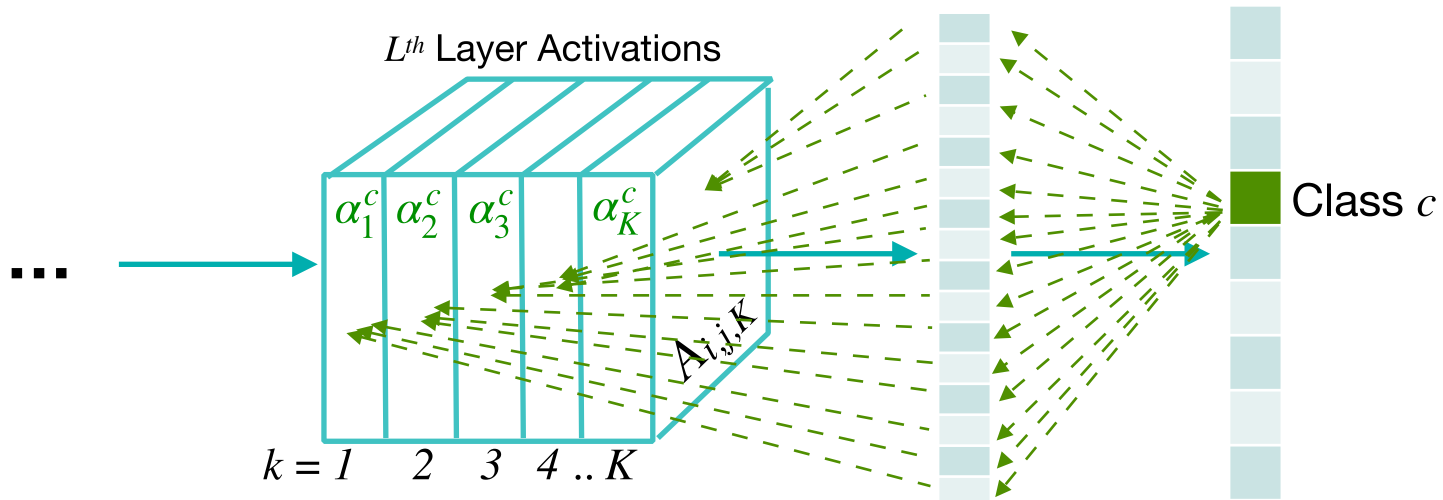
# Class Activation Mapping (CAM)

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

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

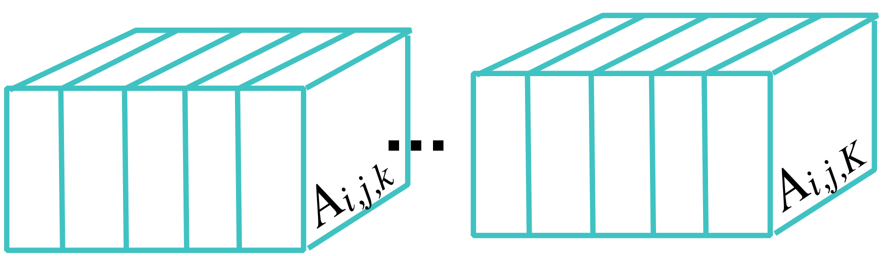
final convolutional layer, $L$, activations
for row, column, channel

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:

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

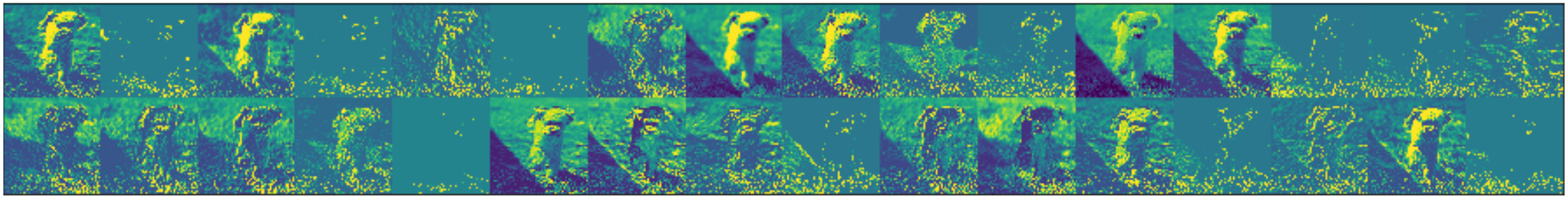
relu activation

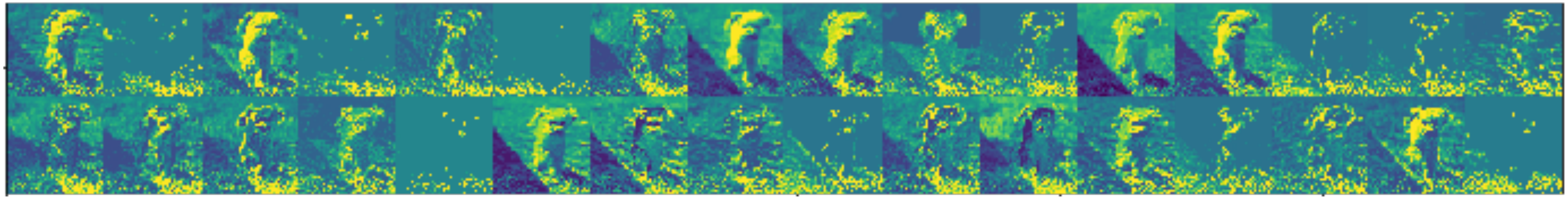



Ramprasath et al., **Grad-CAM:** Visual Explanations from Deep Networks via Gradient-based Localization

# Visualizing ConvNets

## Part Three: Grad-CAM



Follow Along: 04 LectureVisualizingConvnets.ipynb
activation-demo

Lecture Notes for

# Neural Networks and Machine Learning

## CNN Visualization

**Next Time:**
CNN Circuits

**Reading:** OpenAI Circuits