# BST 261: Data Science II
# Lecture 8
## Convolutional Neural Networks (CNNs): Fine-tuning and Visualizing what CNNs Learn

Heather Mattie
Harvard T.H. Chan School of Public Health
Spring 2 2020

# Recipe of the Day!

Blackberry Macarons

# Paper Presentation

Article | Open Access | Published: 20 September 2019

# Deep learning algorithm predicts diabetic retinopathy progression in individual patients

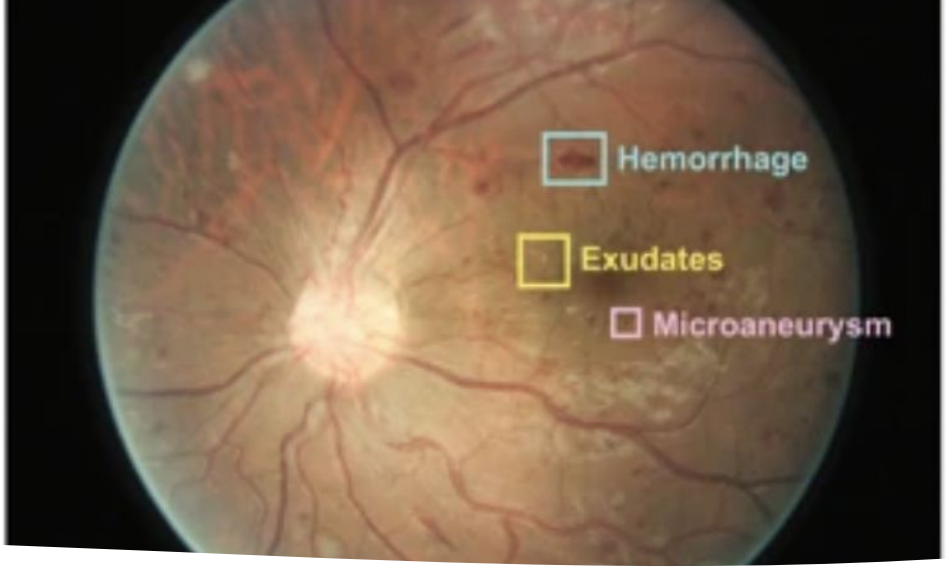
Filippo Arcadu, Fethallah Benmansour, Andreas Maunz, Jeff Willis, Zdenka Haskova & Marco Prunotto

*npj Digital Medicine* **2**, Article number: 92 (2019) | Cite this article



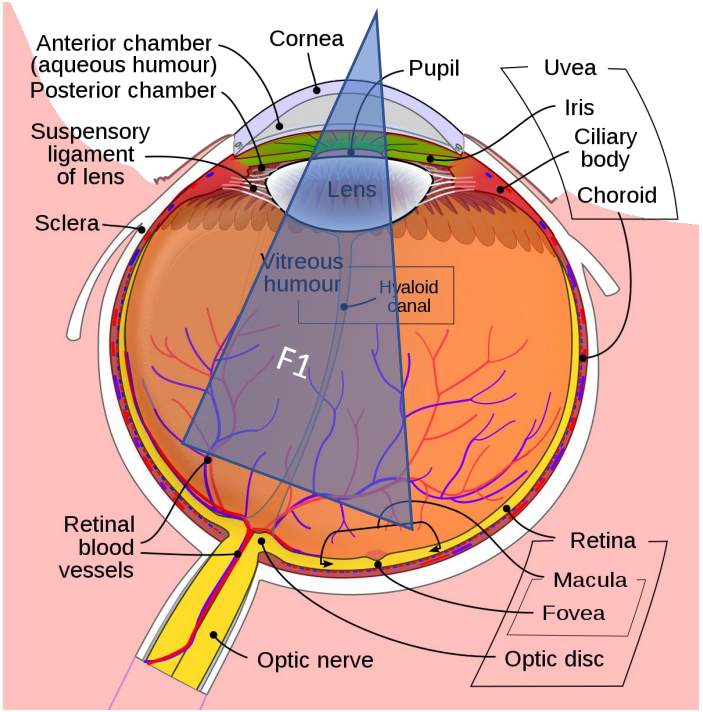
BST 261

Santiago Romero-Brufau

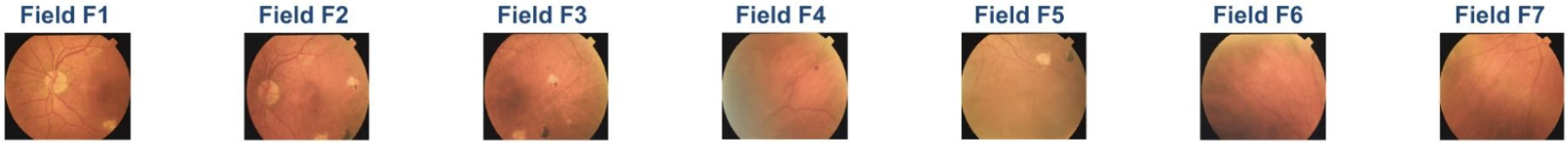



# Introduction

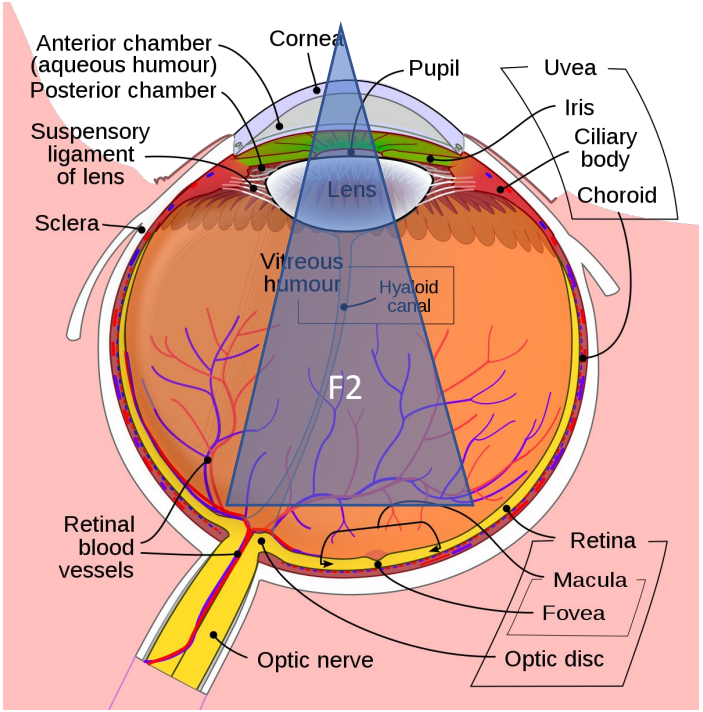
- Diabetic retinopathy often remains undetected until it progresses to an advanced vision-threatening stage.
- There are treatments that are effective, but they rely on early detection.
- Detection is currently based on images of the retina, evaluated by humans, but this is expensive and not easily scalable.

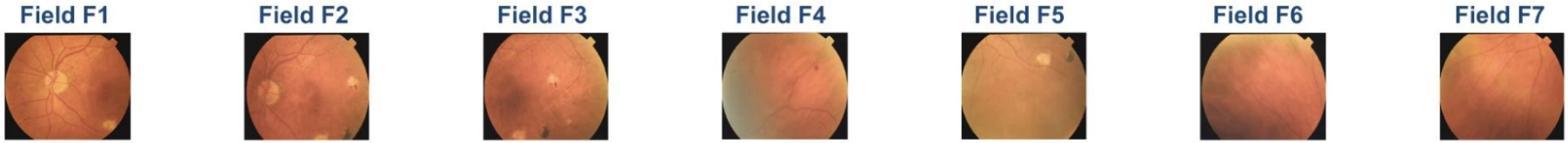



# Introduction: Retinal fields




Patient's 7 CFP fields at baseline

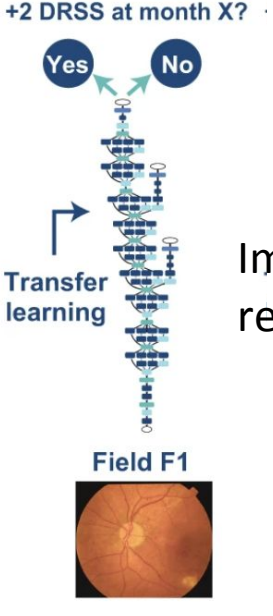



Patient's 7 CFP fields at baseline

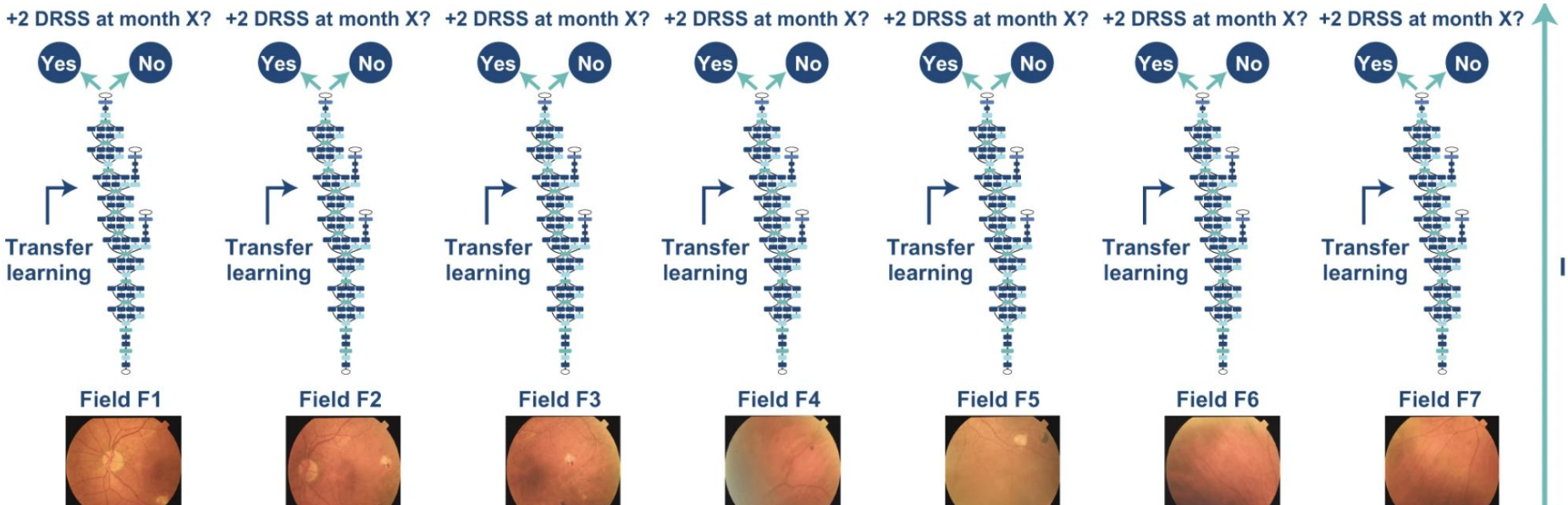
Methods

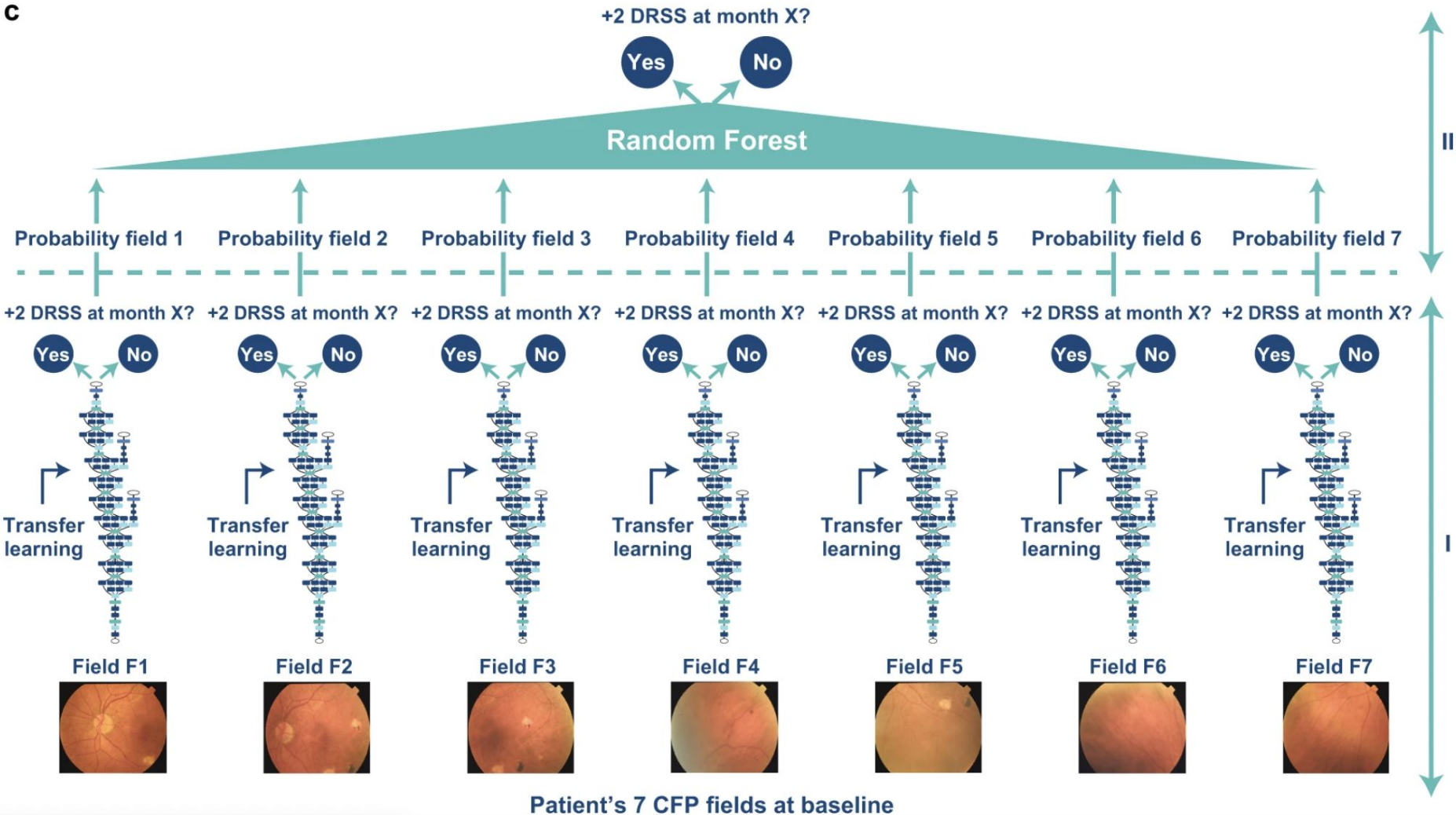
- Data
- Data from clinical trials (high-quality) (683 eyes at 6m, 645 eyes at 2y)
- 7 fields (not common)
- Ground truth: Diabetic Retinopathy Severity Scale (DRSS)



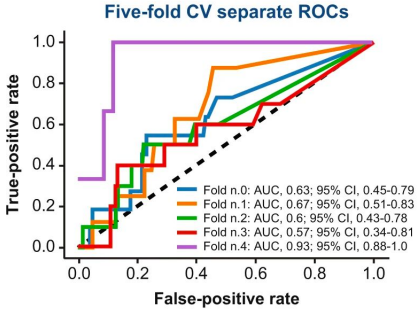

Imagenet weights and trained on the Kaggle DR dataset, then re-trained on their ~600 images from their dataset.

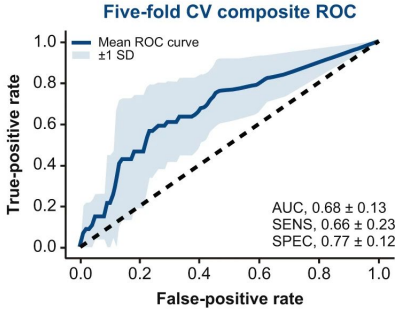
They did that for all 7 fields for each patient-eye



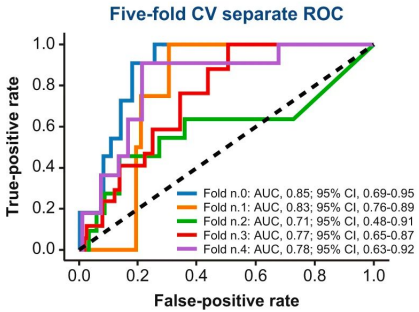

c
+2 DRSS at month X?
Yes
No
Random Forest
II
Probability field 1
Probability field 2
Probability field 3
Probability field 4
Probability field 5
Probability field 6
Probability field 7
+2 DRSS at month X?
+2 DRSS at month X?
+2 DRSS at month X?
+2 DRSS at month X?
+2 DRSS at month X?
+2 DRSS at month X?
+2 DRSS at month X?
Yes
No
Transfer learning
I
Field F1
Field F2
Field F3
Field F4
Field F5
Field F6
Field F7
Patient's 7 CFP fields at baseline

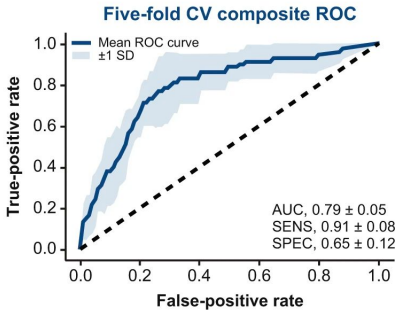
Month 6



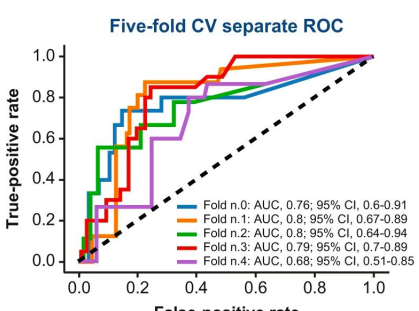



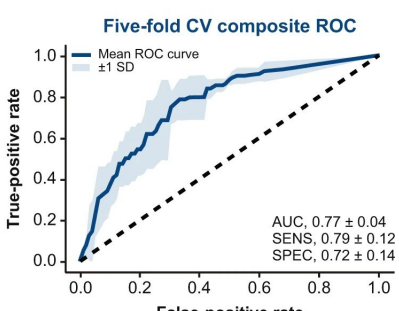

Month 12




Month 24

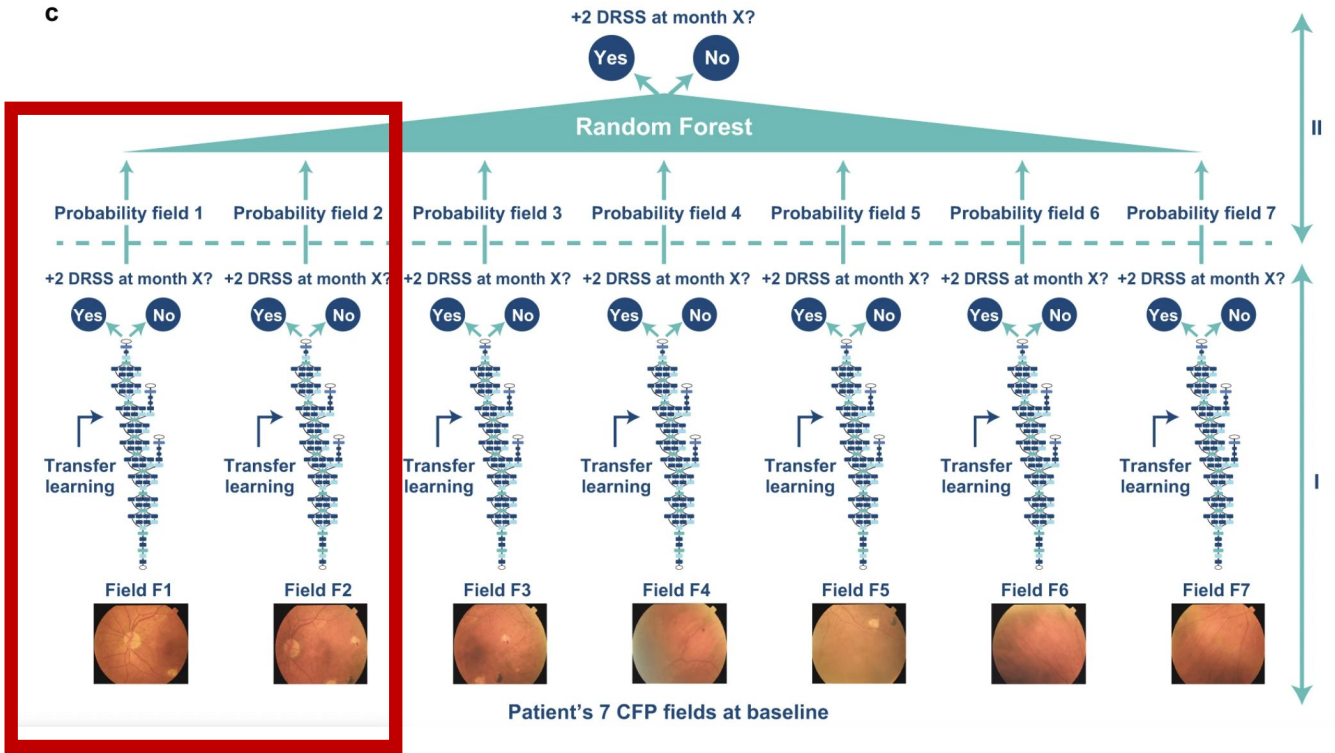


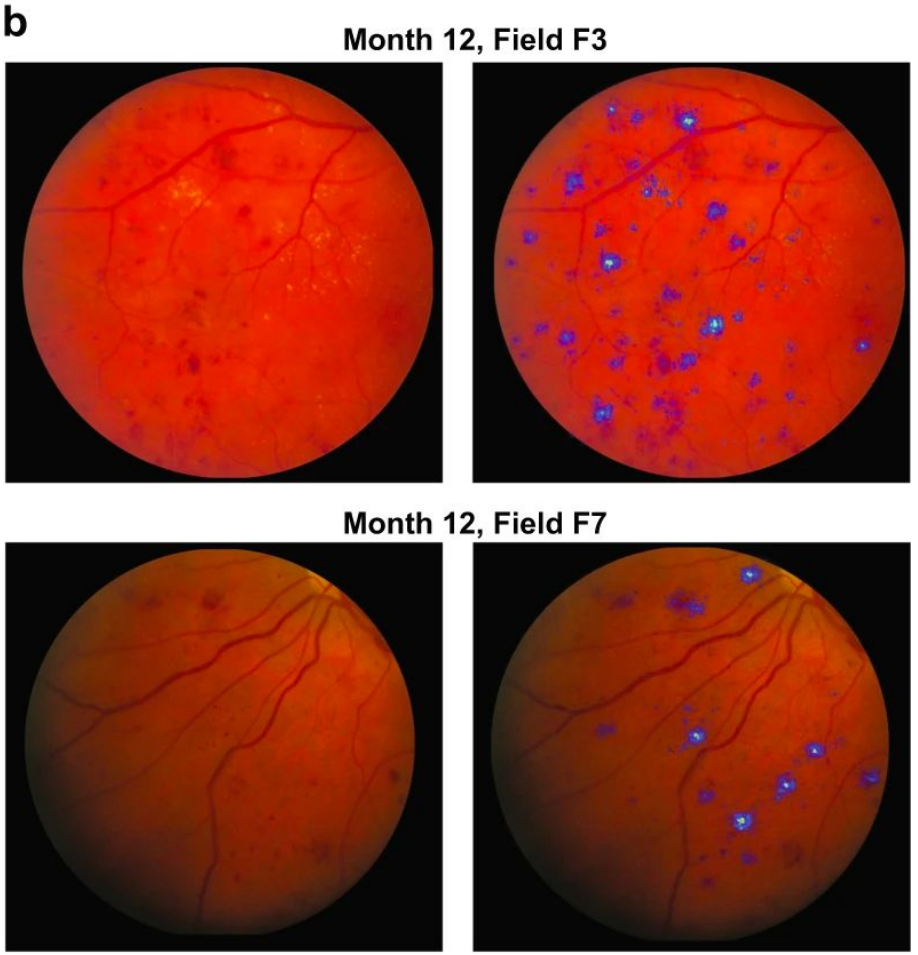
AUC 0.64 with F1 and F2 at 12 months

AUC 0.79 with the full model at 12 months

They also concluded that fields F5 and F7 play a more crucial role in the prediction.



Attribution maps show the model is focusing on exudates, hemorrhages (the spots you see)



Take-home points

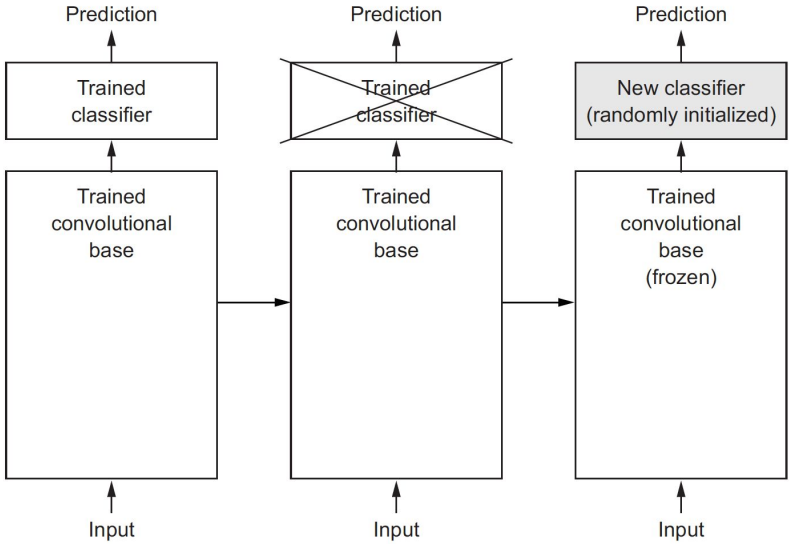
- 1) CNN (w/transfer learning) on each field + Random Forest

- 2) Using the data-based approach, they demonstrated that the areas of the retina that had the most information were different from the ones that were commonly thought.

# Pretrained Networks

# Feature Extraction

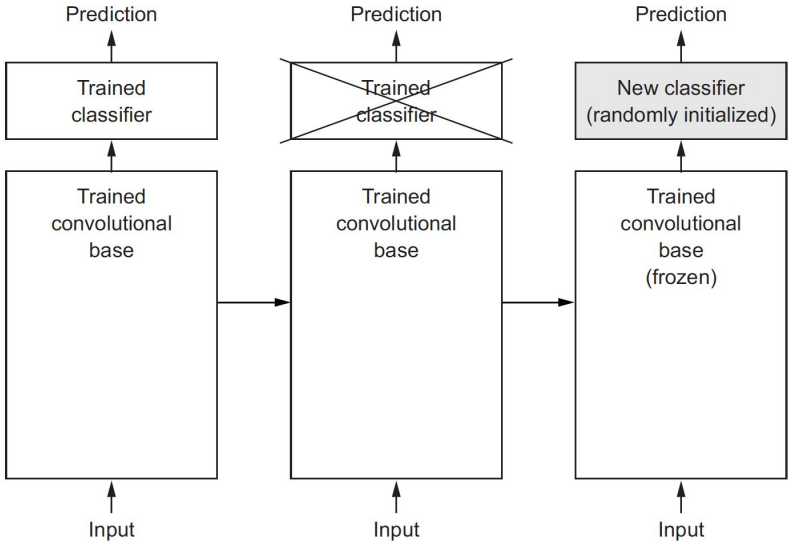
- Consists of using the representations learned by a previous network to extract features from new samples
- These features are then run through a new classifier that is trained from scratch, and predictions are made



# Feature Extraction

- For CNNs, the part of the pretrained network you use is called the **convolutional base**, which contains a series of convolution and pooling layers

- For feature extraction, you keep the convolutional base of the pretrained network, remove the dense / trained classifier layers, and append new dense and classifier layers to the convolutional base



# Feature Extraction with Augmented Data

[Colab notebook]

```
model = tf.keras.models.Sequential([
  conv_base,
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(256, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0
```

We can add the base just like a layer to our network

```
model = tf.keras.models.Sequential([
  conv_base,
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(256, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0
```

```
1 print('This is the number of trainable weights '
2       'before freezing the conv base:', len(model.trainable_weights))
```

```
This is the number of trainable weights before freezing the conv base: 30
```

```
1 conv_base.trainable = False
```

```
1 print('This is the number of trainable weights '
2       'after freezing the conv base:', len(model.trainable_weights))
```

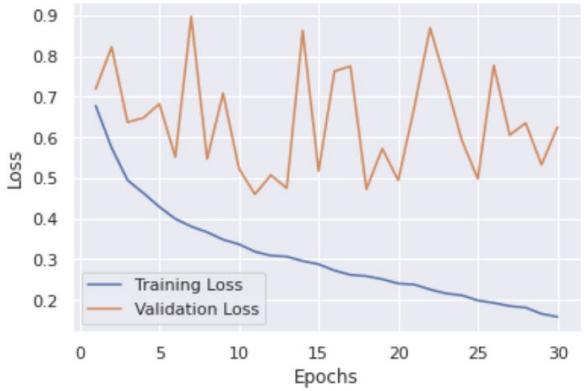
```
This is the number of trainable weights after freezing the conv base: 4
```

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
      rescale=1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 150x150
        target_size=(150, 150),
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=2e-5),
              loss='binary_crossentropy',
              metrics=['accuracy'])


history = model.fit(
      train_generator,
      steps_per_epoch=81,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=22)
```

Note: do not run this code without access to a GPU.

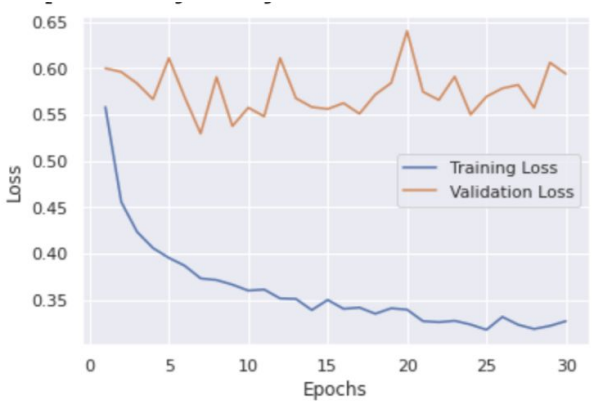
Back to [Colab notebook]

## Original CNN made from scratch with data augmentation





## CNN using pretrained base with data augmentation




# Fine-tuning

# Fine-tuning

- **Fine-tuning** consists of unfreezing a few of the top layers of a frozen model base used for feature extraction, and jointly training both the newly added part of the model (the dense layers used to classify), and these top unfrozen layers
  - This slightly adjusts the more abstract representations of the pretrained model in an effort to make them more relevant for the problem at hand
  - **It is only possible to fine-tune the top layers of the convolutional base**, and only after the added classifier layers have been trained

# Steps

- Add your custom network on top of an obtained pretrained base network
- Freeze the base network
- Train the part you added
- Unfreeze some layers in the base network
- Jointly train both the unfrozen layers and top layers

We did the first 3 steps when we did feature extraction

# Fine-tuning

- In practice it is good to unfreeze 2-3 top layers of the base
- The more layers you unfreeze, the more parameters that need to be trained, and the higher the risk of overfitting (longer to train as well)
- Note that **earlier layers in the base encode more generic, reusable features, and layers higher up encode more specialized features**. Thus, it's more useful to fine-tune layers higher up in the base

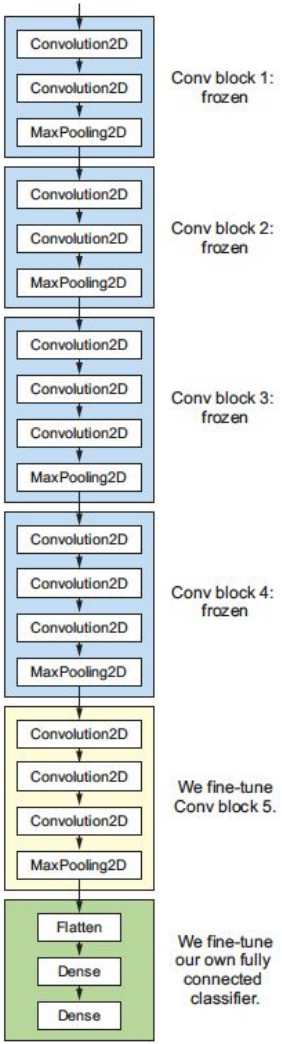



Figure 5.19 Fine-tuning the last convolutional block of the VGG16 network

We only unfreeze and fine-tune the last block of layers.

```
conv_base.trainable = True

set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

We need to say which pretrained blocks (and layers) should be kept frozen (make untrainable) and which one we want to unfreeze (make trainable).

```
model.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.RMSprop(lr=1e-5),
              metrics=['accuracy'])

history = model.fit(
      train_generator,
      steps_per_epoch=100,
      epochs=100,
      validation_data=validation_generator,
      validation_steps=50)
```

# Visualizing what CNNs Learn

# Visualizing What CNNs Learn

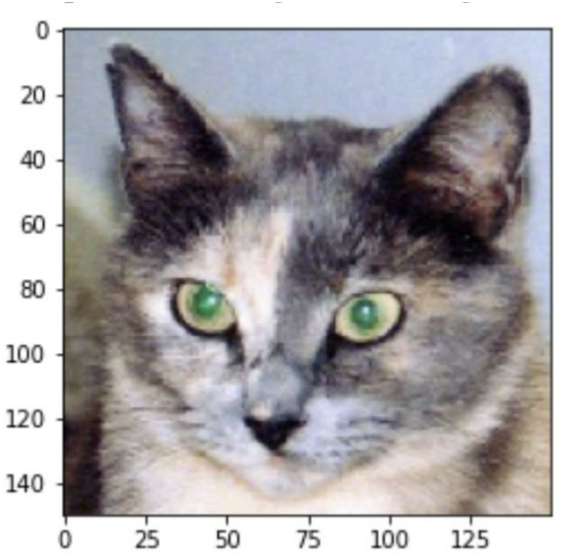
- It is possible to visualize and interpret the learned representations of your CNN
- 3 of the most useful visualizations are:
  - **Visualizing intermediate activations**
    - Useful for understanding how successive layers transform their input and getting an idea of the meaning of individual filters
  - **Visualizing filters**
    - Useful for understanding what visual pattern or concept each filter in a CNN is receptive to
  - **Visualizing heatmaps** of class activations in an image
    - Useful for understanding which parts of an image were identified as belonging to a given class

# Visualizing Intermediate Outputs

# Visualizing Intermediate Outputs

- Colab notebook
- Display the feature maps that are output by various convolution and pooling layers
- You should look at each channel separately

```
1 img_path = os.path.join(test_dir, 'cats/cat.1700.jpg')
2
3 # We preprocess the image into a 4D tensor
4 from keras.preprocessing import image
5 import numpy as np
6
7 img = image.load_img(img_path, target_size=(150, 150))
8 img_tensor = image.img_to_array(img)
9 img_tensor = np.expand_dims(img_tensor, axis=0)
10 # Remember that the model was trained on inputs
11 # that were preprocessed in the following way:
12 img_tensor /= 255.
13
14 # Its shape is (1, 150, 150, 3)
15 print(img_tensor.shape)
```



# Visualizing Intermediate Outputs

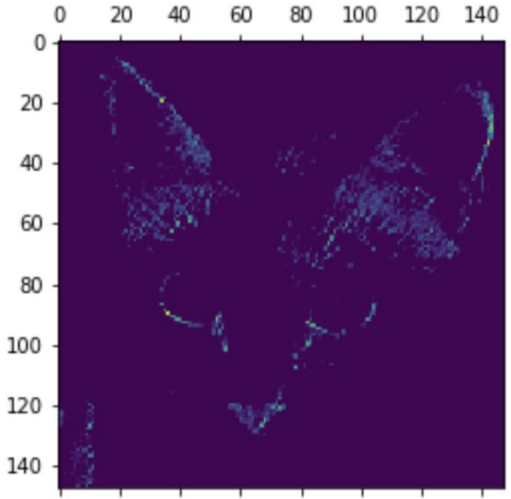
```
1 from keras import models
2
3 # Extracts the outputs of the top 8 layers:
4 layer_outputs = [layer.output for layer in model.layers[:8]]
5 # Creates a model that will return these outputs, given the model input:
6 activation_model = tf.keras.models.Model(inputs=model.input, outputs=layer_outputs)
7
8 # This will return a list of 5 Numpy arrays:
9 # one array per layer activation
10 activations = activation_model.predict(img_tensor)
11
12 first_layer_activation = activations[0]
13
14 import matplotlib.pyplot as plt
15
16 plt.matshow(first_layer_activation[0, :, :, 11], cmap = 'viridis')
17 plt.show()
```

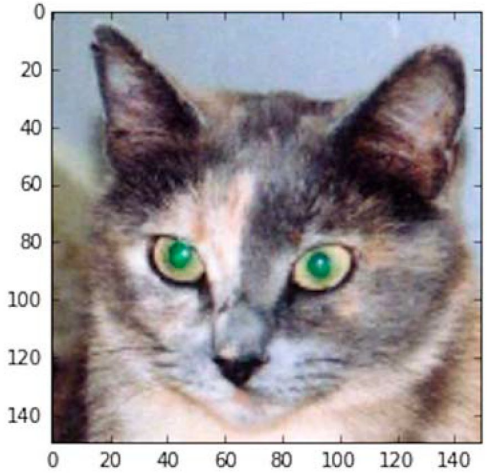
This will save the outputs or “activations” for each filter in every layer

Let’s look at the filters in the first layer

We’ll visualize what patterns this filter is picking up

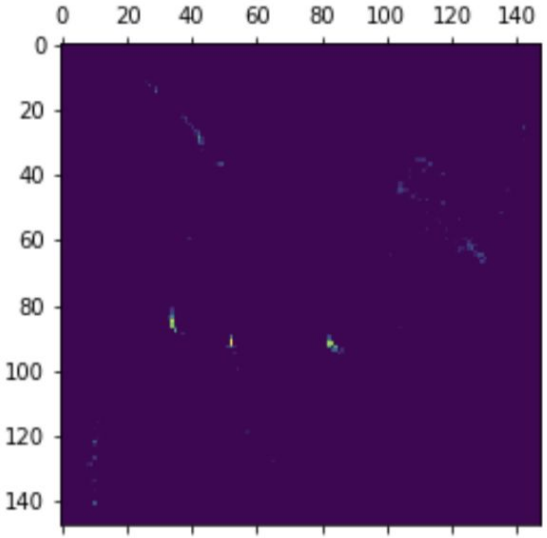
# Visualizing Intermediate Outputs

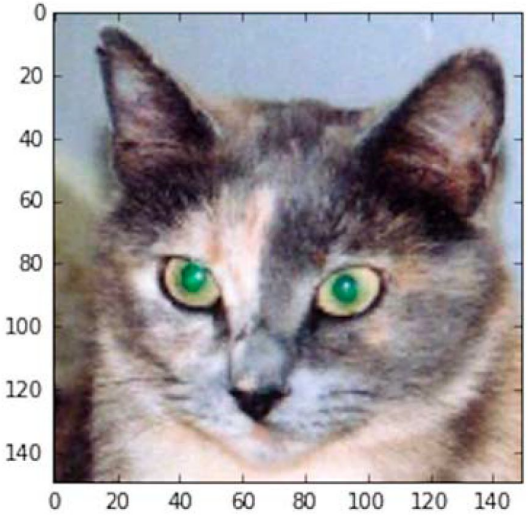



Diagonal/rounded edges filter?

# Visualizing Intermediate Outputs

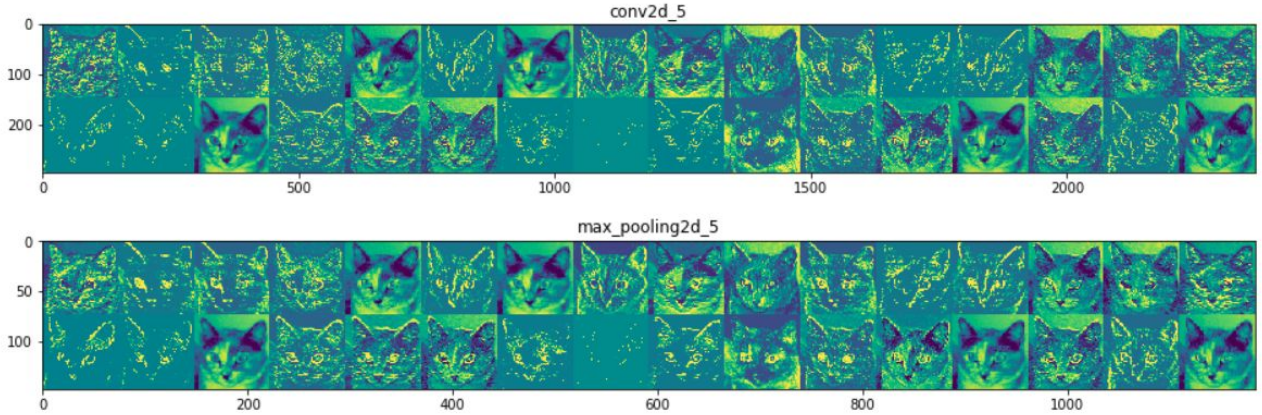


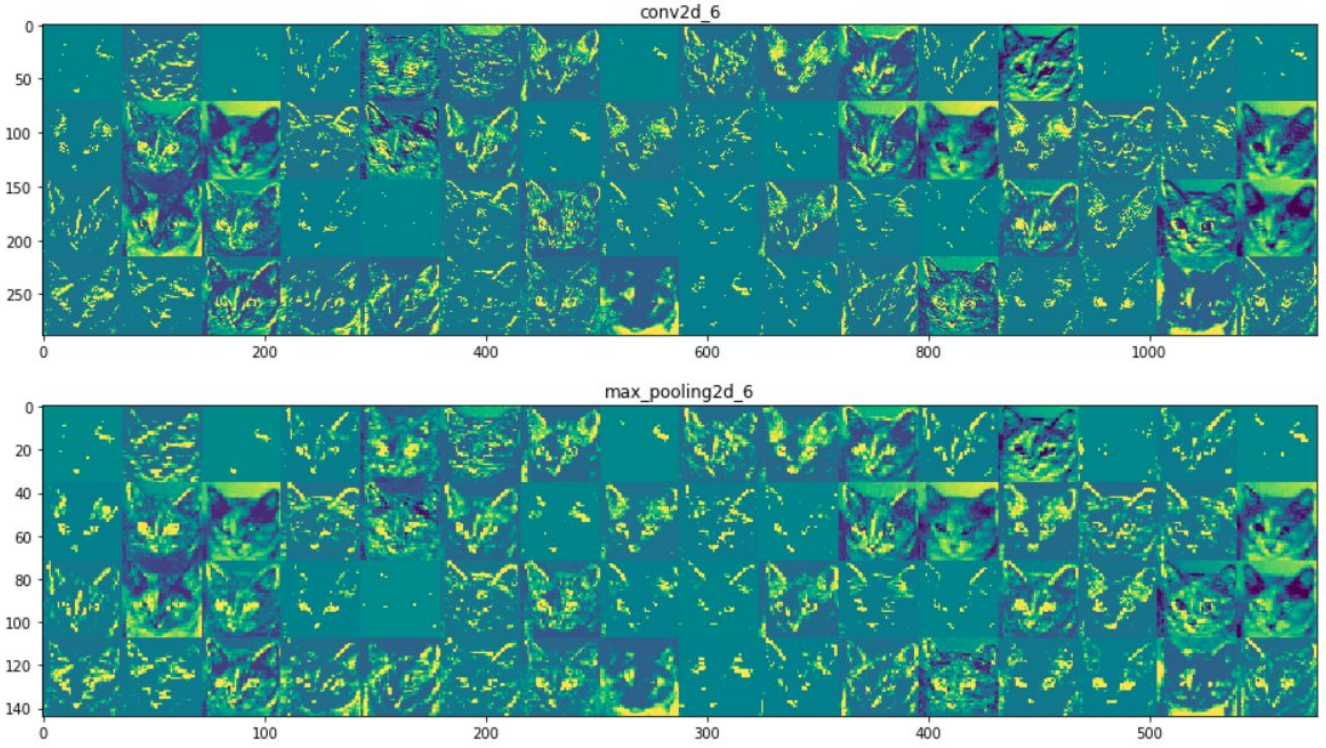

“Green dots” filter?

# Visualizing Intermediate Outputs

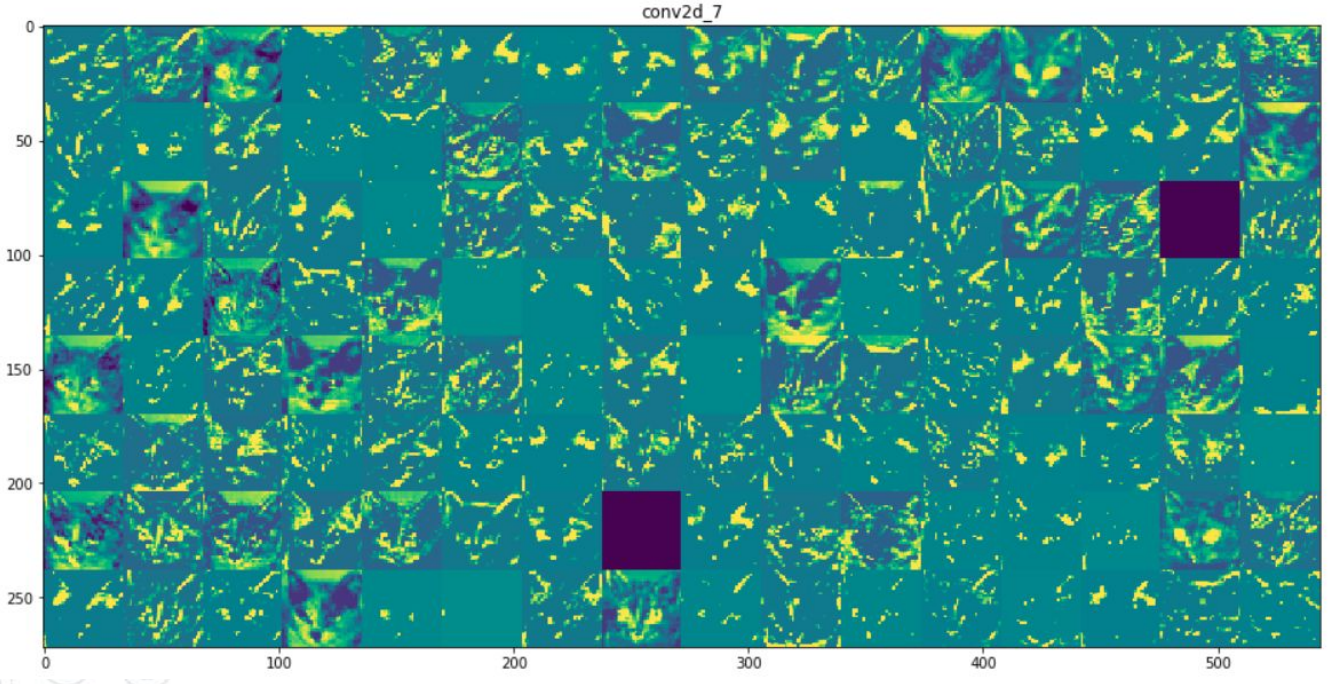
- We can also look at what pattern each filter in every layer is picking up on



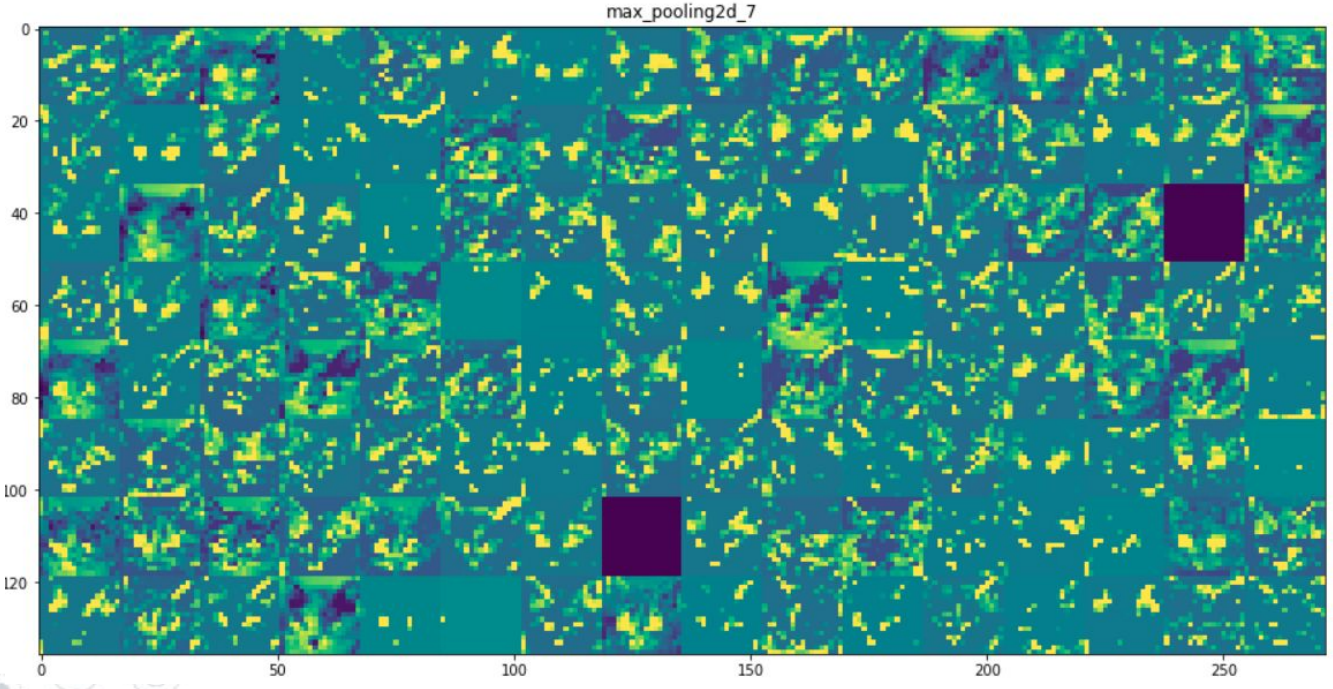
# Visualizing Intermediate Outputs



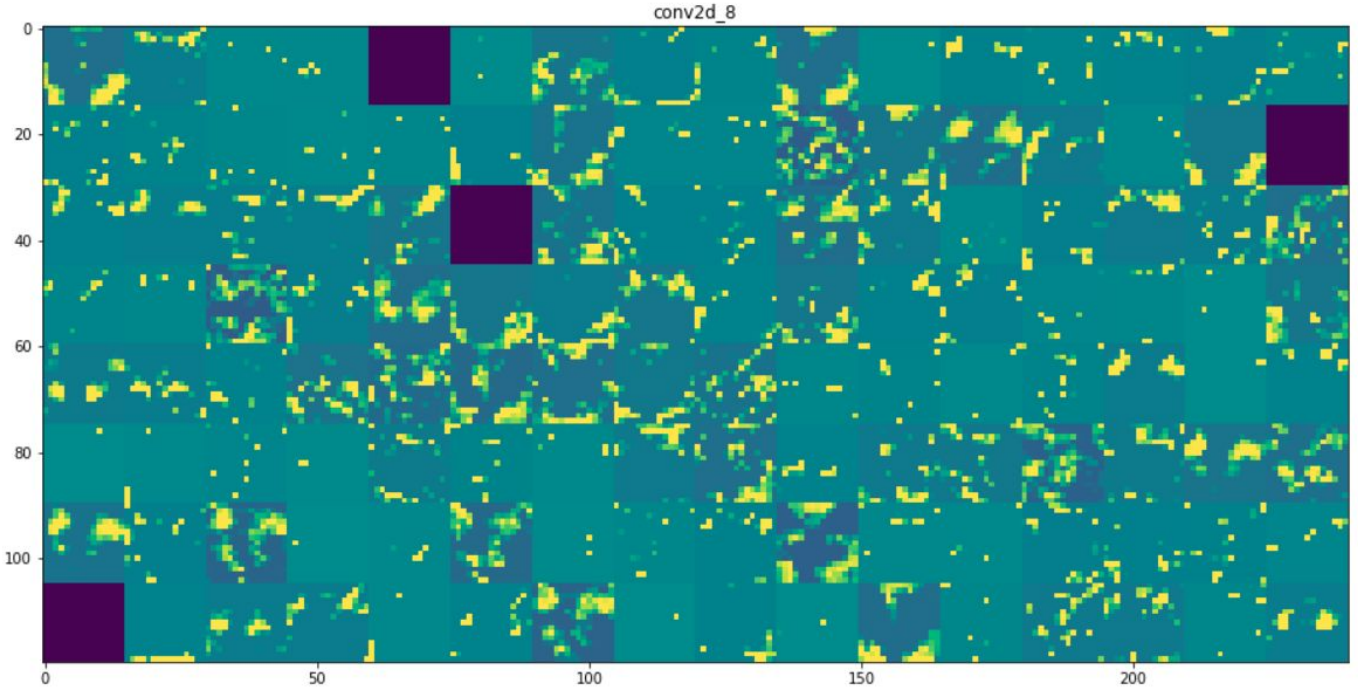
# Visualizing Intermediate Outputs



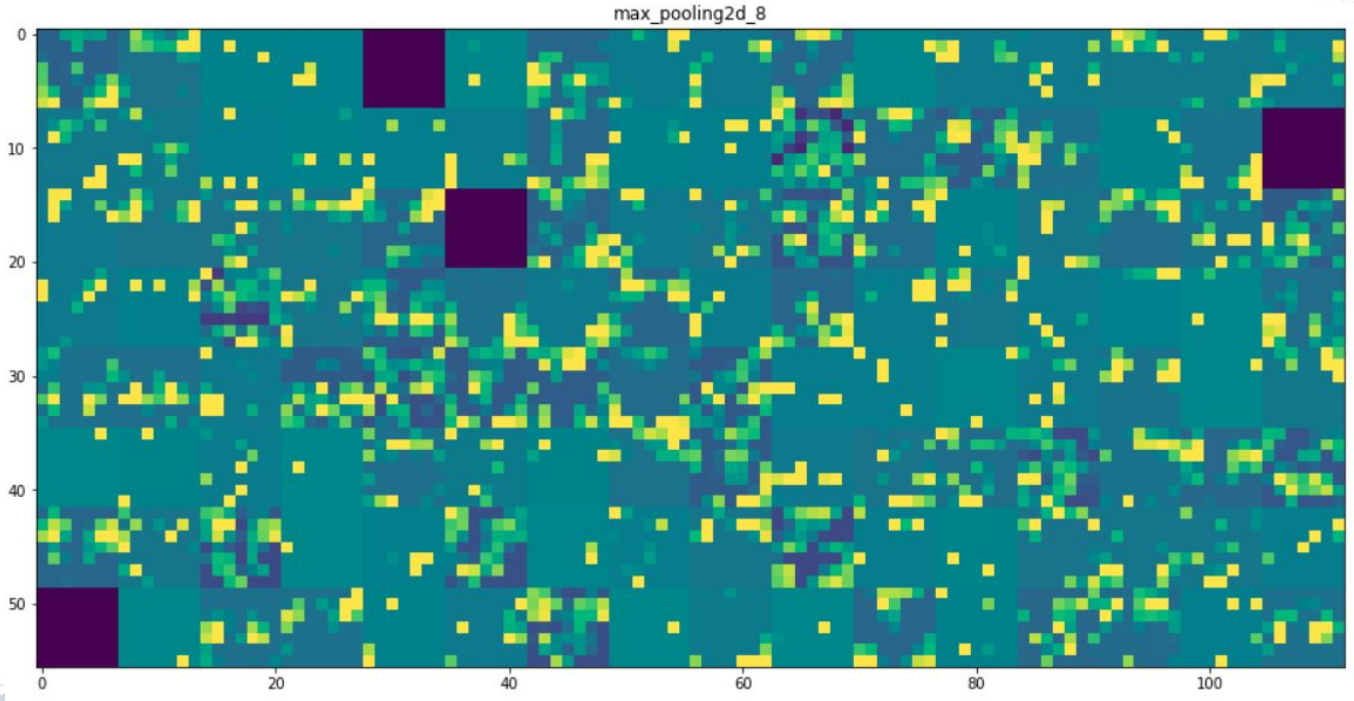
# Visualizing Intermediate Outputs



# Visualizing Intermediate Outputs



# Visualizing Intermediate Outputs



# Visualizing Intermediate Outputs

- The first layer acts as a collection of edge detectors
- The later layers contain more abstract activations that are less visually interpretable
- Deeper layers carry less information about visual contents of the image, and more information related to the class of the image

# Visualizing Intermediate Outputs

- The sparsity of the activations increases with the depth of the layer

- Blank activations mean the pattern encoded by that filter isn't found in the input image

# Visualizing Filters

# Visualizing Filters

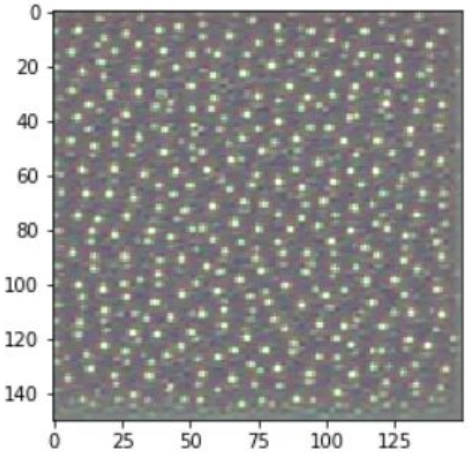
- Shows the visual pattern that each filter is meant to respond to
- This is done with gradient ascent in input space: applying gradient descent to the value of the input image to maximize the response of a specific filter, starting with a blank input image
- The resulting image will be one that the chosen filter is maximally responsive to

# Visualizing Filters

- Steps:
  - Build a loss function that maximizes the value of a given filter in a given convolution layer
  - Use stochastic gradient descent to adjust the values of the input image in order to maximize the activation value

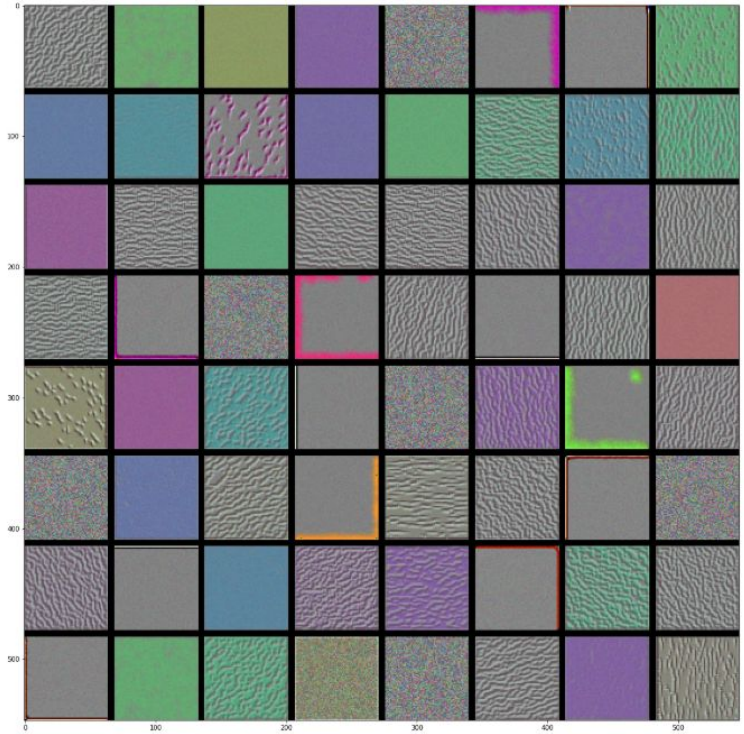
# Visualizing Filters

The “polka dots” filter



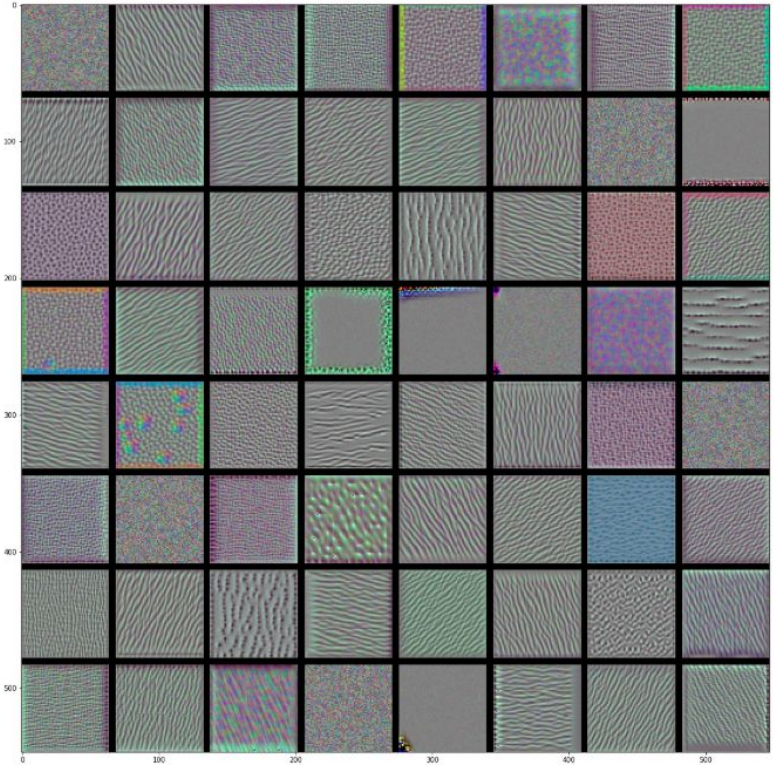
# Visualizing Filters

- Filters from the 1st convolution block



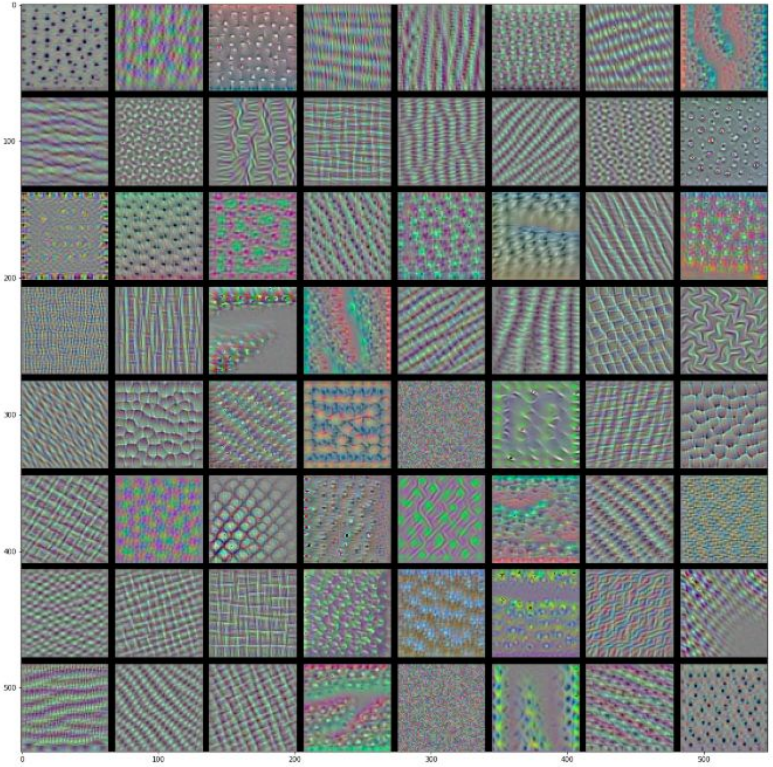
# Visualizing Filters

- Filters from the second convolution block



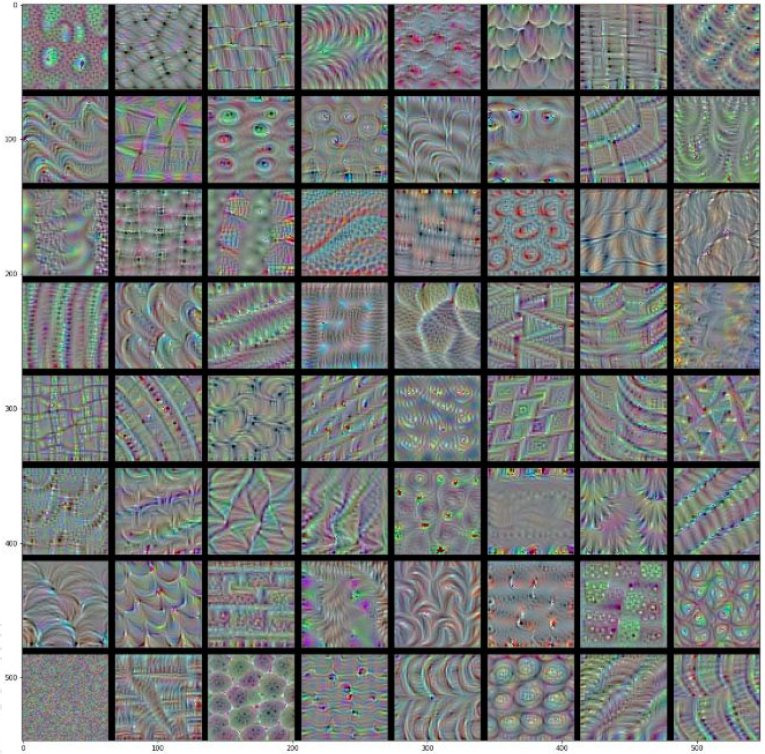
# Visualizing Filters

- Filters from the third convolution block



# Visualizing Filters

- Filters from the fourth convolution block



# Visualizing Filters

- The filters get increasingly complex and refined as you go deeper in the model
- The filters from the first layer encode single directional edges and colors
- The next set of filters encode simple textures made from combinations of edges and colors
- The filters in later layers resemble textures found in natural images - eyes, feathers, leaves, etc.

# Visualizing Heatmaps of Class Activation

# Visualizing Heatmaps of Class Activation

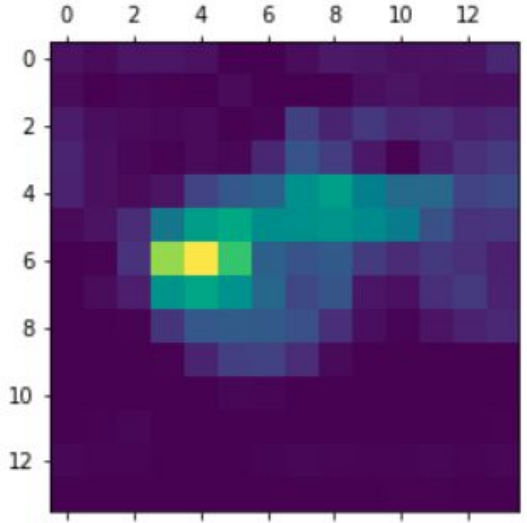
- Great for understanding which parts of an image led the network to its final classification
- Helpful for debugging the decision process
- This also allows you to locate specific objects in an image
- Called class activation map (CAM) visualization
- A class activation heatmap is a 2D grid of scores associated with a specific output class, computed for every location in an input image, indicating how important each location is with respect to the class under consideration

# Visualizing Heatmaps of Class Activation

- When we run this image of African elephants through the VGG16 network, the following are the top 3 predictions:
  - African elephant (with 92.5% probability)
  - Tusker (with 7% probability)
  - Indian elephant (with 0.4% probability)

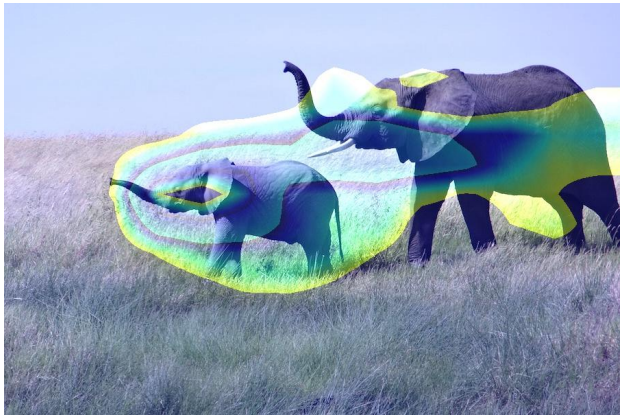
# Visualizing Heatmaps of Class Activation

- Lighter colors (yellow, green) correspond to greater activation and darker colors (blue, purple) to less or no activation, allowing us to see which parts of the image were used for the classification




# Visualizing Heatmaps of Class Activation

- We can then overlap these activations with the original image to see exactly what and where in the image was used in classification



# Visualizing Heatmaps of Class Activation

- When we run this image of a Turkish Shepherd through the VGG16 network, the following are the top 3 predictions:

# Visualizing Heatmaps of Class Activation

- When we run this image of a Turkish Shepherd through the VGG16 network, the following are the top 3 predictions:
  - Saluki (with 65.9% probability)

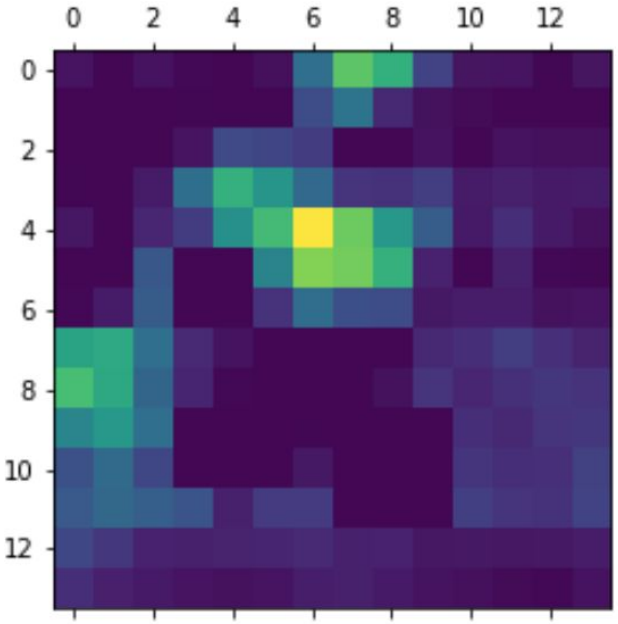
# Visualizing Heatmaps of Class Activation

- When we run this image of a Turkish Shepherd through the VGG16 network, the following are the top 3 predictions:
  - Saluki (with 65.9% probability)
  - Whippet (with 6.3% probability)

# Visualizing Heatmaps of Class Activation

- When we run this image of a Turkish Shepherd through the VGG16 network, the following are the top 3 predictions:
  - Saluki (with 65.9% probability)
  - Whippet (with 6.3% probability)
  - Labrador retriever (with 3.9% probability)

# Visualizing Heatmaps of Class Activation




# Visualizing Heatmaps of Class Activation

# Visualizing Heatmaps of Class Activation

- When we run this image of a Harvard gate through the VGG16 network, the following are the top 3 predictions:

# Visualizing Heatmaps of Class Activation

- When we run this image of a Harvard gate through the VGG16 network, the following are the top 3 predictions:
  - Prison (with 41.3% probability)

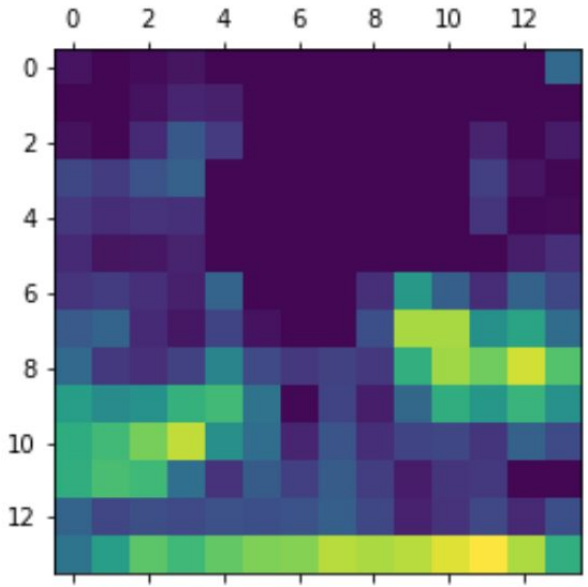
# Visualizing Heatmaps of Class Activation

- When we run this image of a Harvard gate through the VGG16 network, the following are the top 3 predictions:
  - Prison (with 41.3% probability)
  - Fire screen (with 10.6% probability)

# Visualizing Heatmaps of Class Activation

- When we run this image of a Harvard gate through the VGG16 network, the following are the top 3 predictions:
  - Prison (with 41.3% probability)
  - Fire screen (with 10.6% probability)
  - Monastery (with 7.7% probability)

# Visualizing Heatmaps of Class Activation




# Visualizing Heatmaps of Class Activation