MACHINE LEARNING

HOW TO GDPR AND EXPLAINING AUTOMATED DECISIONS

BJÖRN GENFORS
NIKLAS ANTONČIĆ

CADEC 2020.01.23 & 2020.01.29 | CALLISTAENTERPRISE.SE

CALLISTA

AGENDA

- Legal stuff
  - The briefest GDPR walkthrough ever
  - 20 months with GDPR - what has happened?
  - GDPR in ML, tips and tricks
  - GDPR requirement: explaining automated decisions

- The fun stuff
  - Explaining an automated decision

WHY?

GDPR ~~GONE~~ IN 60 SECONDS

CALLISTA

GDPR, WHAT IS IT?

- EU regulation in effect since May 2018
- Regulates use of personal data

GDPR TERMINOLOGY

- **Personal data** ("personuppgift")
  - Special categories
- **Processing** ("personuppgiftsbehandling")
- **Controller** ("personuppgiftsansvarig")

GDPR, GENERAL PRINCIPLES

- Lawful and transparent
- Accurate and fair
- Minimisation
- Purpose limitation
- Privacy by design and default

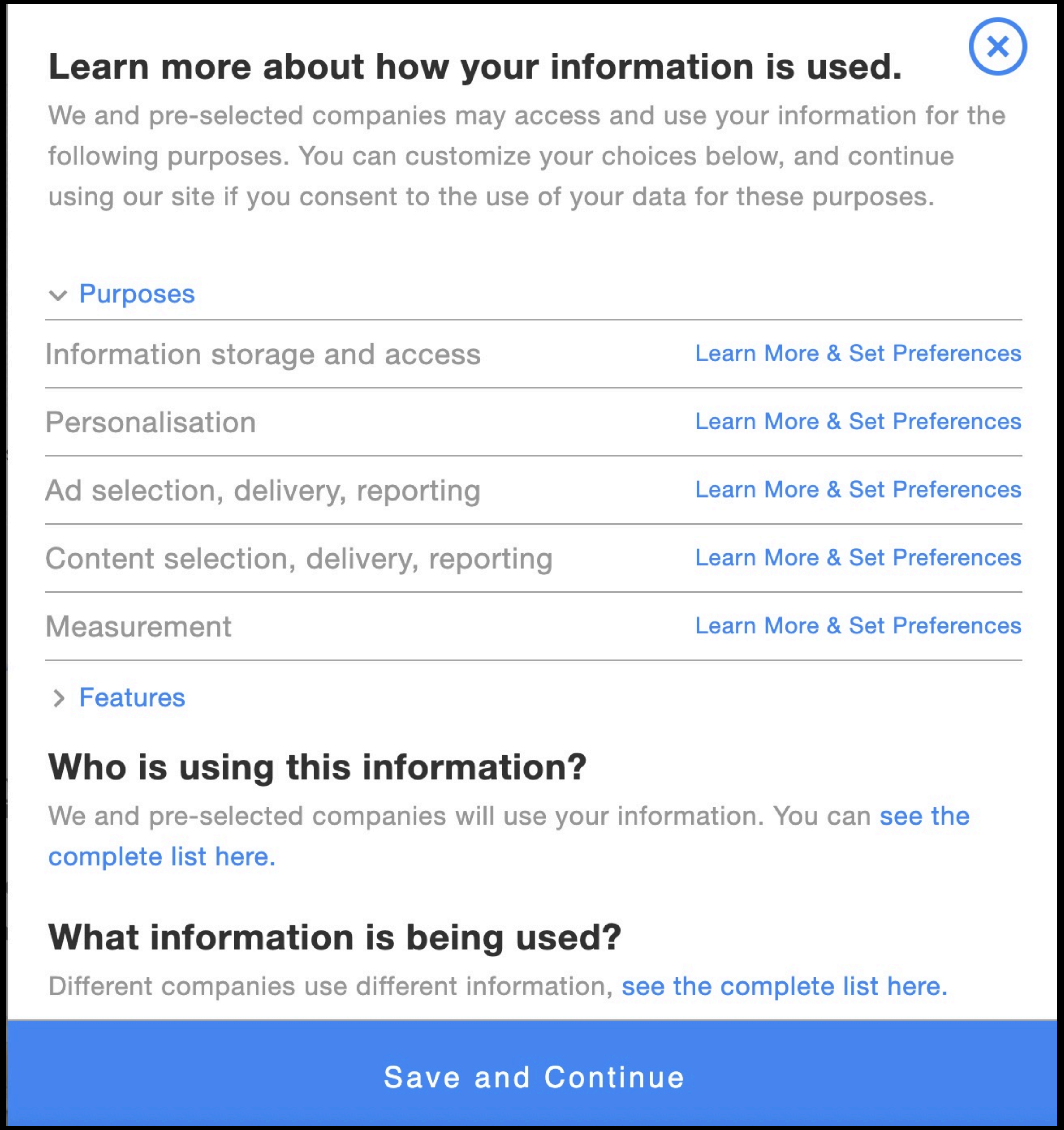
GDPR SO FAR

GDPR EFFECTS

Before

Source: https://imgflip.com/i/2au4ju

After



ON A MORE SERIOUS NOTE

- ≥189 GDPR fines to date (enforcementtracker.com, 2020-01-23)
  - Most common reasons for fines
    - » Insufficient legal basis for data processing
    - » Insufficient technical and organisational measures to ensure information security
    - » Insufficient fulfilment of data subjects rights
    - » Non-compliance with general data processing principles
  - 2 Swedish cases, 1 regarding facial recognition
  - No ML specific fines

GDPR IN ML

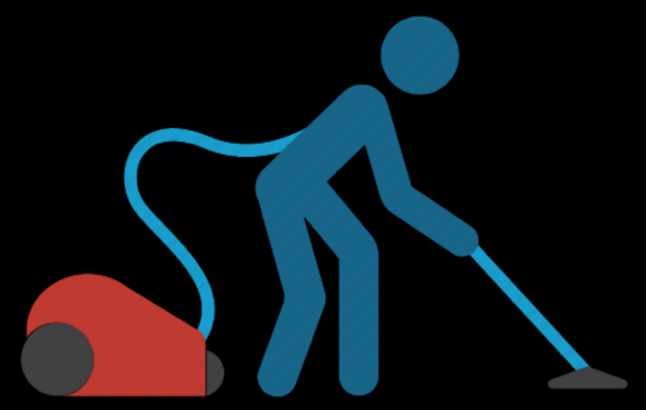
THE PROCESS


BUSINESS TARGET
ACQUIRE RAW DATA
PRE PROCESS
SELECT MODEL
TRIM OR CHANGE MODEL
TRAIN
IMPLEMENT
FINAL HYPOTHESIS
VALIDATE RESULT
FINAL HYPOTHESIS

THE PROCESS

%

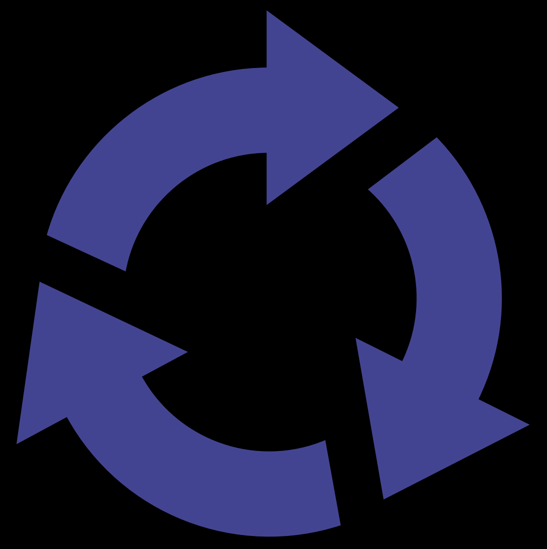
BUSINESS TARGET

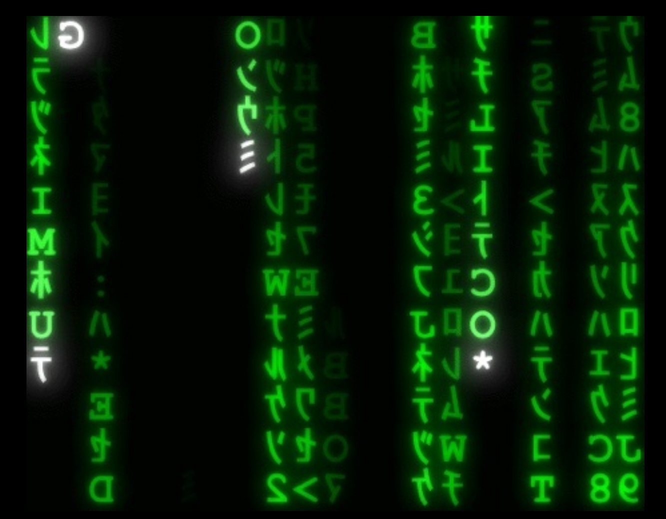


TRAINING DATA

SELECT MODEL

TRIM OR CHANGE MODEL





TRAIN

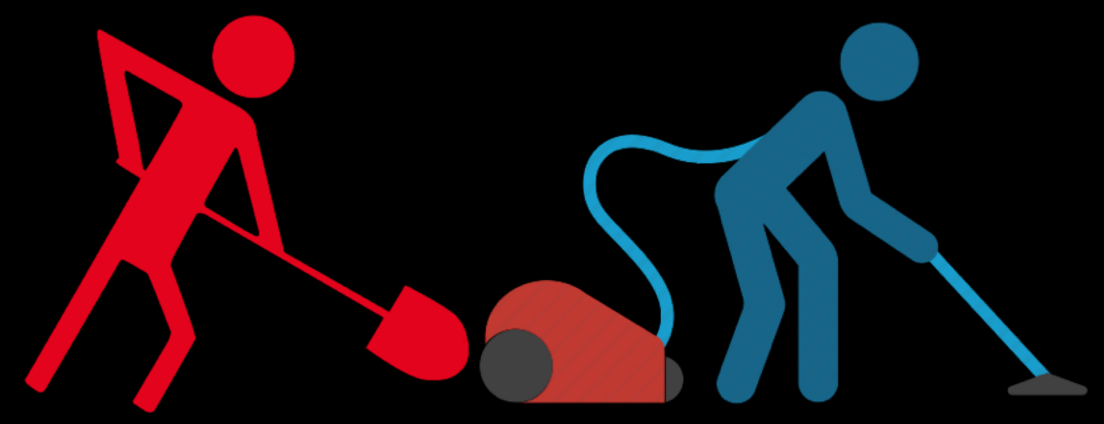
IMPLEMENT

FINAL HYPOTHESIS

VALIDATE RESULT

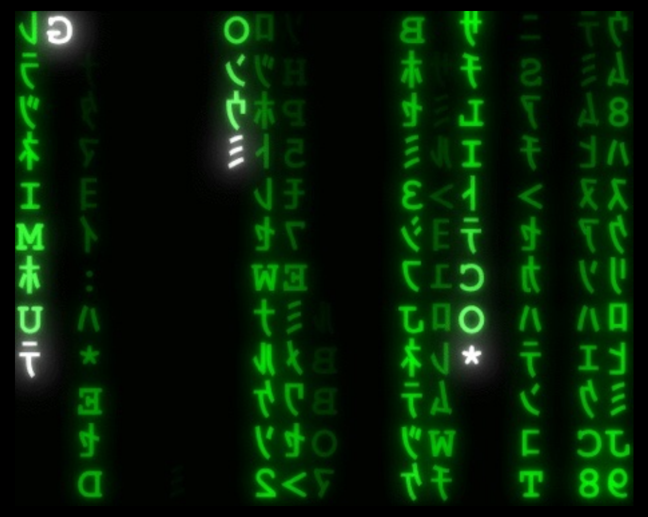
FINAL HYPOTHESIS

TRAINING DATA



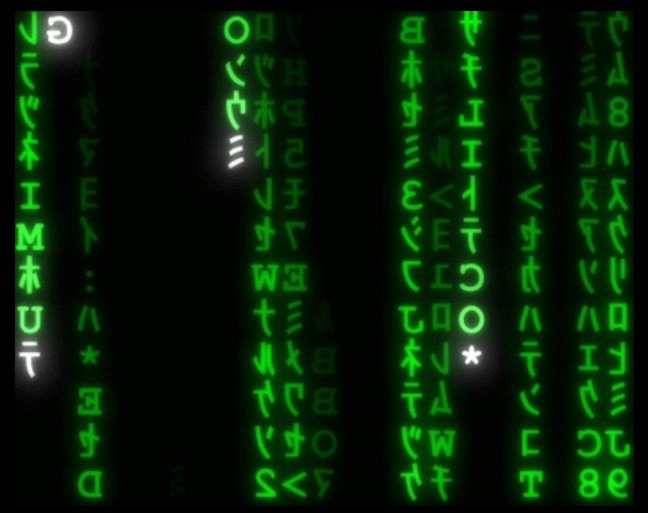
- Federated Learning
  - In essence: no need to collect training data
- Matrix Capsules
  - Special type of CNN. Reduced need for training data
- Generative Adversarial Networks (GAN)/Variational autoencoders (VAE)
  - Generates training data - reduces need for actual data

TRAIN



- Differential Privacy
  - Adding noise to algorithm
- Transfer Learning
  - Reduce need for training data
- Homomorphic encryption
  - Run algorithm on encrypted data

HOMOMORPHIC ENCRYPTION



ElGamal

In the ElGamal cryptosystem, in a cyclic group G of order q with generator g, if the public key is (G, q, g, h), where $h = g^x$, and x is the secret key, then the encryption of a message m is $\mathcal{E}(m) = (g^r, m \cdot h^r)$, for some random $r \in \{0, \ldots, q-1\}$. The homomorphic property is then

$$\begin{aligned}\mathcal{E}(m_1) \cdot \mathcal{E}(m_2) &= (g^{r_1}, m_1 \cdot h^{r_1})(g^{r_2}, m_2 \cdot h^{r_2}) \\ &= (g^{r_1+r_2}, (m_1 \cdot m_2)h^{r_1+r_2}) \\ &= \mathcal{E}(m_1 \cdot m_2).\end{aligned}$$

Goldwasser–Micali

In the Goldwasser–Micali cryptosystem, if the public key is the modulus n and quadratic non-residue x, then the encryption of a bit b is $\mathcal{E}(b) = x^b r^2 \mod n$, for some random $r \in \{0, \ldots, n-1\}$. The homomorphic property is then

$$\begin{aligned}\mathcal{E}(b_1) \cdot \mathcal{E}(b_2) &= x^{b_1} r_1^2 x^{b_2} r_2^2 \mod n \\ &= x^{b_1+b_2}(r_1 r_2)^2 \mod n \\ &= \mathcal{E}(b_1 \oplus b_2).\end{aligned}$$

where $\oplus$ denotes addition modulo 2, (i.e. exclusive-or).

Benaloh

In the Benaloh cryptosystem, if the public key is the modulus n and the base g with a blocksize of c, then the encryption of a message m is $\mathcal{E}(m) = g^m r^c \mod n$, for some random $r \in \{0, \ldots, n-1\}$. The homomorphic property is then

$$\begin{aligned}\mathcal{E}(m_1) \cdot \mathcal{E}(m_2) &= (g^{m_1} r_1^c)(g^{m_2} r_2^c) \mod n \\ &= g^{m_1+m_2}(r_1 r_2)^c \mod n \\ &= \mathcal{E}(m_1 + m_2).\end{aligned}$$

Paillier

In the Paillier cryptosystem, if the public key is the modulus n and the base g, then the encryption of a message m is $\mathcal{E}(m) = g^m r^n \mod n^2$, for some random $r \in \{0, \ldots, n-1\}$. The homomorphic property is then

$$\begin{aligned}\mathcal{E}(m_1) \cdot \mathcal{E}(m_2) &= (g^{m_1} r_1^n)(g^{m_2} r_2^n) \mod n^2 \\ &= g^{m_1+m_2}(r_1 r_2)^n \mod n^2 \\ &= \mathcal{E}(m_1 + m_2).\end{aligned}$$

Source: https://en.wikipedia.org/wiki/Homomorphic_encryption

VALIDATE RESULT

IMPLEMENT

- Solely automated decisions of significant effect banned (Article 22)
  - Some exceptions mentioned
  - Almost full ban regarding special categories personal data

- If allowed:
  - Individual's right to information (Article 13-14)
  - Individual's right to explanation? (Article 22)

ARTICLE 22

- Contains safeguards against automated decision making
  - Right to express views
  - Right to contest decision
  - Obtaining human intervention

A RIGHT TO EXPLANATION?

- No?
  - Not explicitly stated in GDPR articles
  - No court rulings
- Yes?
  - No court rulings
  - Recital 71
  - Implicit in Article 22

LÄNKLISTA - GDPR

- Riktlinjer om automatiserat beslutsfattande: https://www.datainspektionen.se/globalassets/dokument/riktlinjer-om-automatiserat-individuellt-beslutsfattande-och-profilering.pdf
- SOU 2018:25 – Juridik som stöd för förvaltningens digitalisering (ffa kap. 7.6) https://www.regeringen.se/rattsliga-dokument/statens-offentliga-utredningar/2018/03/sou-201825/
- Differential privacy and PATE: http://www.cleverhans.io/privacy/2018/04/29/privacy-and-machine-learning.html
- Variational autoencoders: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73
- Generative Adversarial Networks: https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

VISUALIZING AND EXPLAINING DEEP MODELS

NIKLAS ANTONČIĆ

CADEC 2020.01.23 & 2020.01.29 | CALLISTAENTERPRISE.SE

CALLISTA

WHY DO WE CARE ABOUT DEEP MODELS?

- Fantastic machines with super human accuracy
- Easily accessible
- For simpler models it is mildly intuitive what is going on
- For complex models, including DeepLearning, it is not

THE NEED FOR EXPLANATION

- GDPR article 22

THE NEED FOR EXPLANATION

- GDPR article 22
- Not everything is GDPR

SvD SvD.se

"Obehörig algoritm tar beslut i socialtjänsten" | SvD

DEBATT. Socialsekreterare fattar normalt beslut om försörjningsstöd. Men i Trelleborgs kommun är det istället en algoritm som sköter detta – men det går inte att få svar på hur den jobbar. Därför JO-anmäler vi nu Trelleborgs kommun, skriver företrädare för Akademikerförbundet SSR. (109 kB)

THE NEED FOR EXPLANATION

- GDPR article 22
- Not everything is GDPR
- Business acceptance


NyTeknik

Premium / Automation / Digitalisering / Energi / Fordon / Startup / Ingenjörskarriär / Lediga jo

ANNONS

DIGITALISERING

Google Artificiell intelligens

Googles ai bättre än läkare på att hitta bröstcancer

2020-01-02 10:54 Av: Ania Obminska 11 kommentarer

THE NEED FOR EXPLANATION

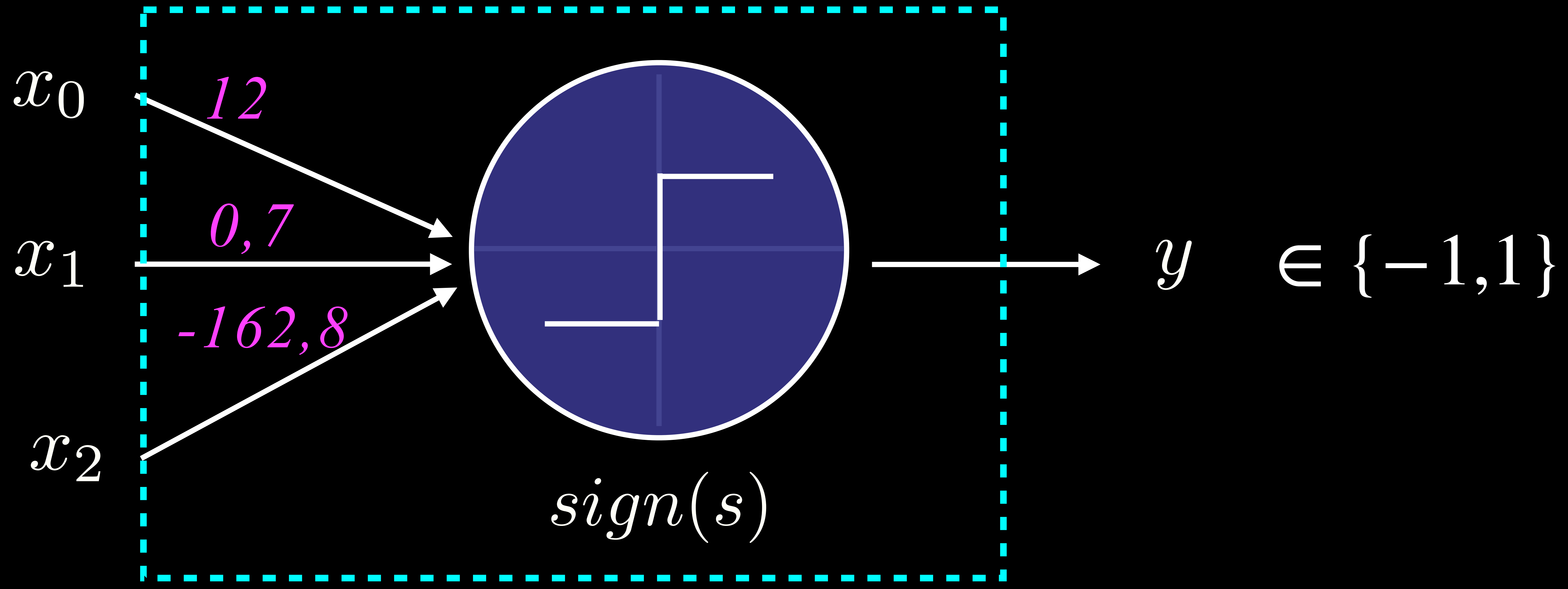
- GDPR article 22
- Not everything is GDPR
- Business acceptance
- Model improvement

LINEAR MODEL - CREDIT APROVAL EXAMPLE

$$w_0 Age + w_1 Income - w_2 Debt \begin{cases} > 0 \Rightarrow Yes \\ < 0 \Rightarrow No \end{cases}$$

TRAIN THE MODEL ON HISTORICAL DATA

LINEAR MODEL - PREDICTER, TRAINED MODEL



$$12Age + 0{,}7Income - 162{,}8Debt = -245, \quad No$$

COMPLEX PROBLEMS - COMPLEX MODELS

- What is this?

COMPLEX PROBLEMS - COMPLEX MODELS

- What is this?

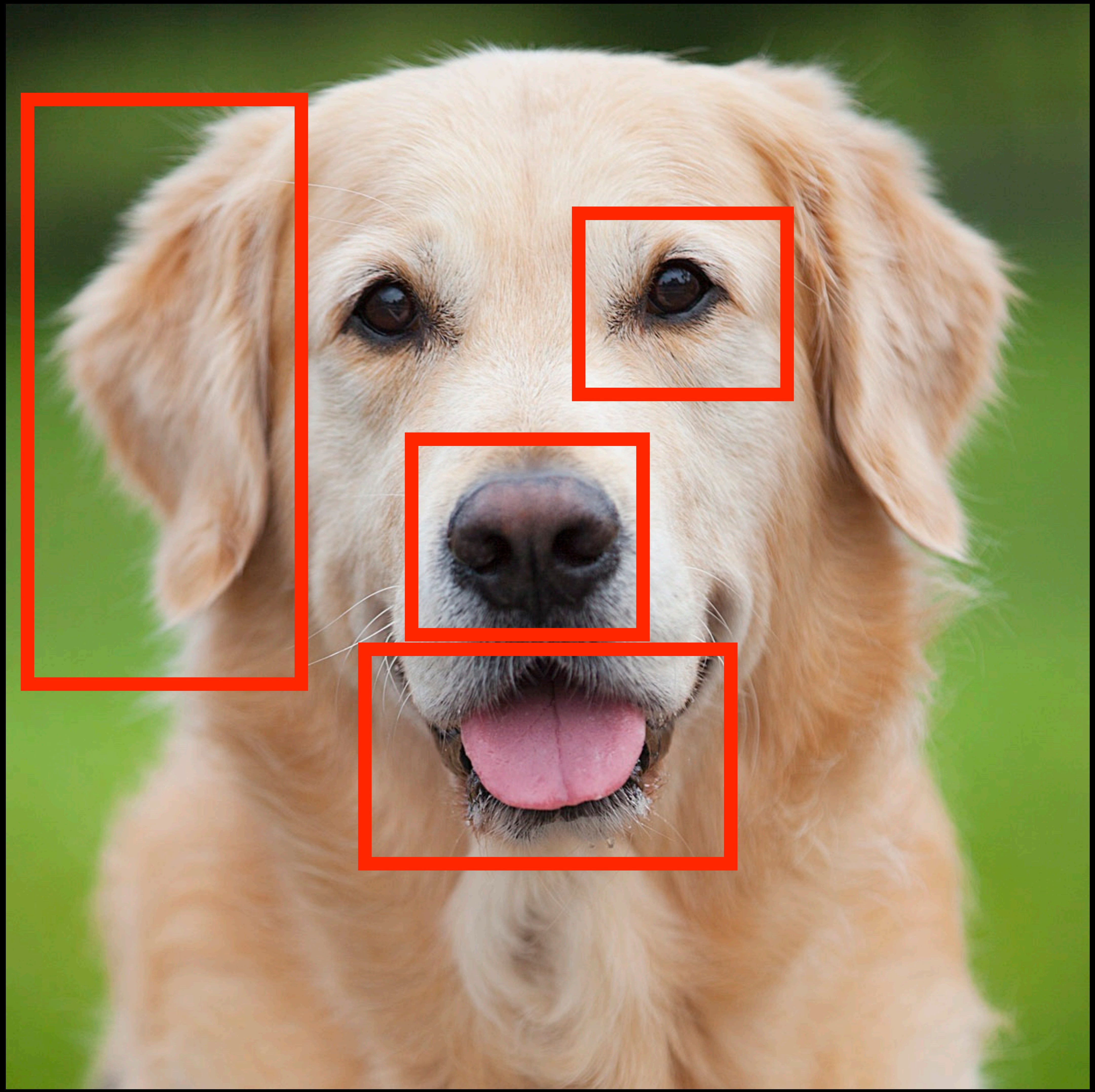
- Image classification problem

COMPLEX PROBLEMS - COMPLEX MODELS

- What is this?
- Image classification problem
- What are the features?

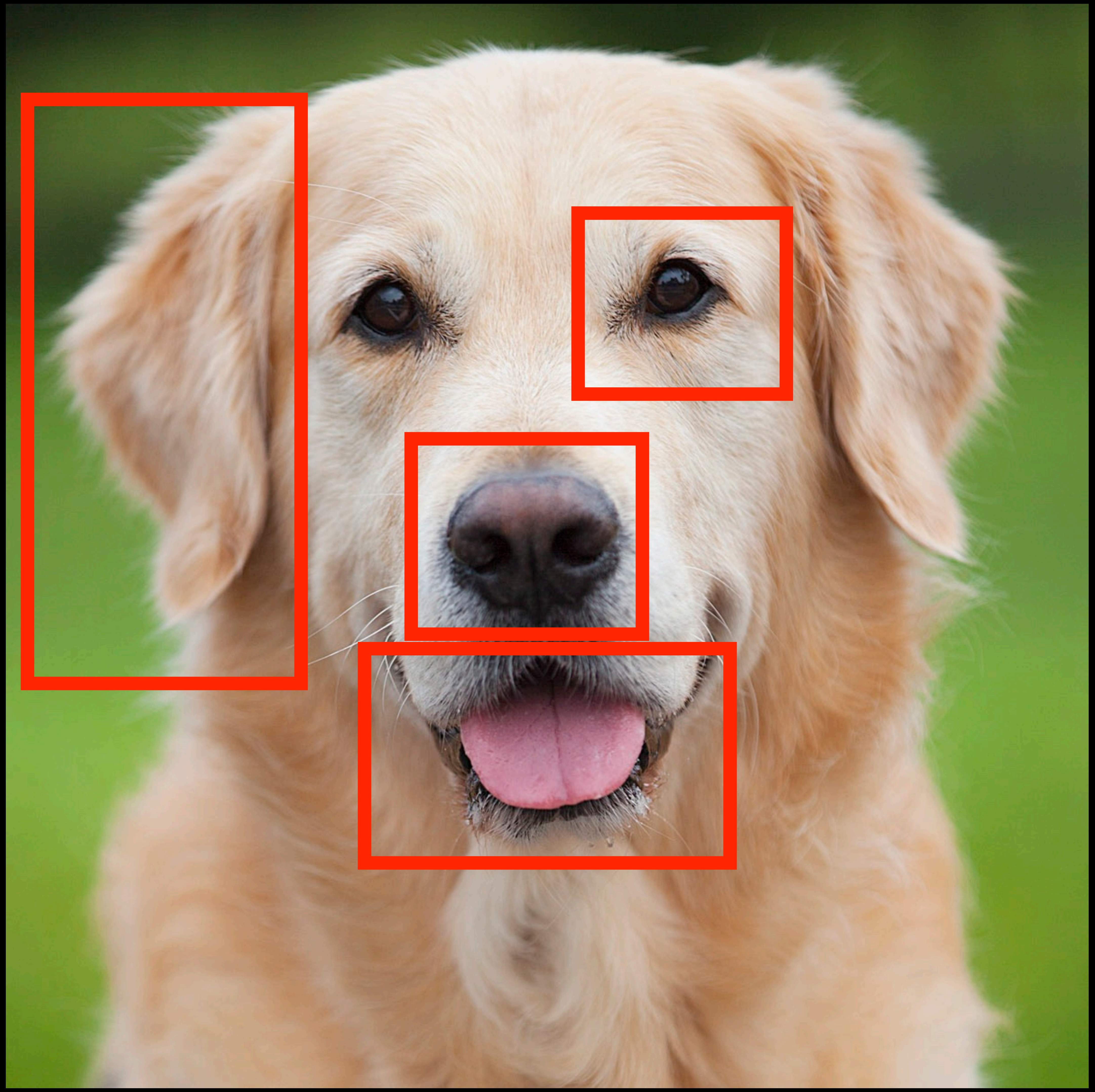
COMPLEX PROBLEMS - COMPLEX MODELS

- What is this?
- Image classification problem
- What are the features?



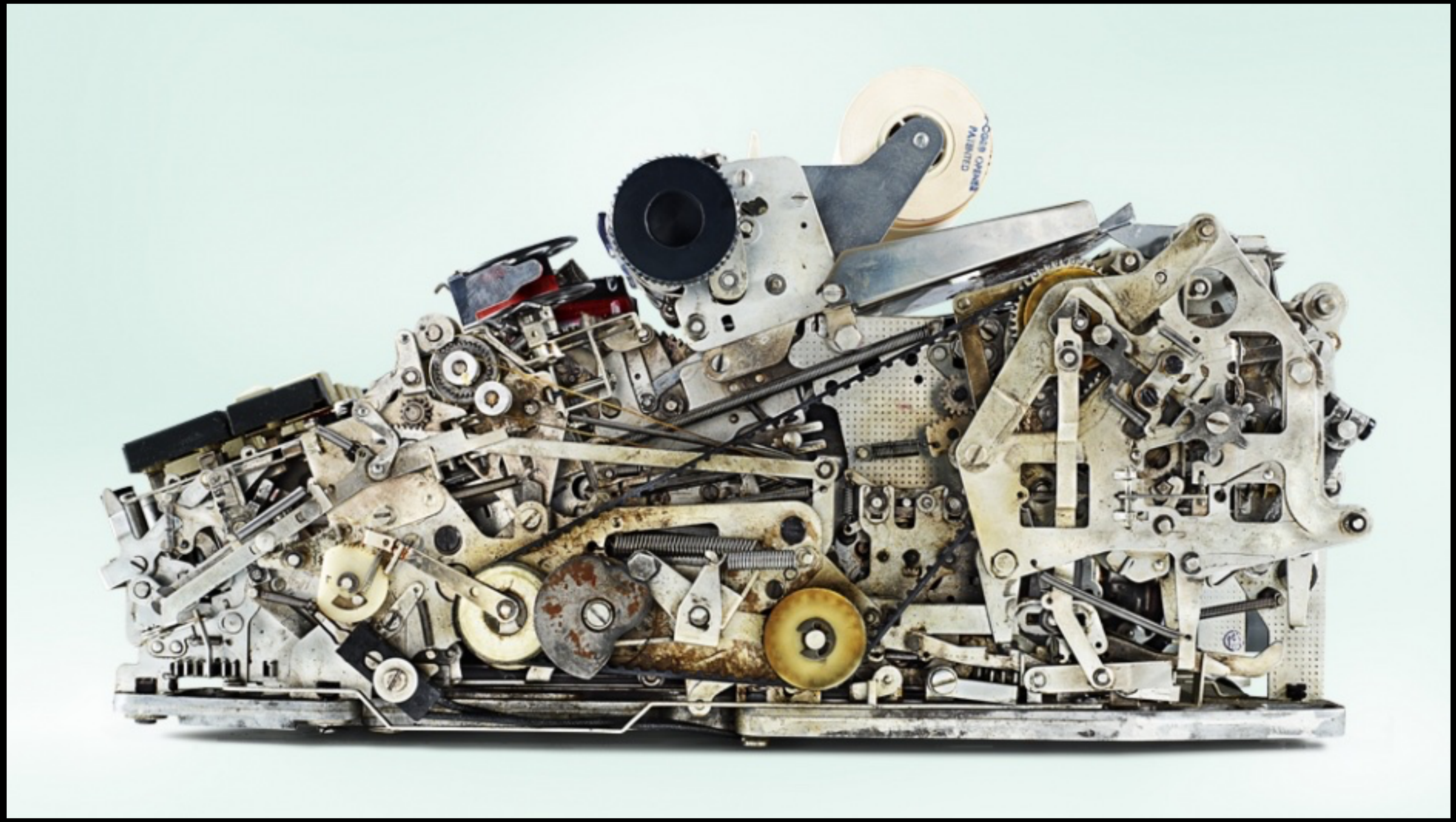
COMPLEX PROBLEMS - COMPLEX MODELS

- What is this?
- Image classification problem
- What are the features?
- Hard to describe mathematically…



COMPLEX PROBLEMS

- Lets try a pre-trained Deep Learning state-of-the-art model, freely available on internet, and trained on 1,2 million images in 1000 categories!



NINE LINES OF CODE

is_this_a_dog.py > ...

```python
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
import numpy as np

model = VGG16(weights='imagenet')

img_path = 'dog.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

predictions = model.predict(x)

for item in decode_predictions(predictions, top=3)[0]:
    print(item[1],item[2])
```

NINE LINES OF CODE

is_this_a_dog.py > ...

```python
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
import numpy as np

model = VGG16(weights='imagenet')

img_path = 'dog.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

predictions = model.predict(x)

for item in decode_predictions(predictions, top=3)[0]:
    print(item[1],item[2])

```

NINE LINES OF CODE

is_this_a_dog.py > ...

```python
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
import numpy as np

model = VGG16(weights='imagenet')

img_path = 'dog.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

predictions = model.predict(x)

for item in decode_predictions(predictions, top=3)[0]:
    print(item[1],item[2])
```

NINE LINES OF CODE

is_this_a_dog.py > ...

```python
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
import numpy as np

model = VGG16(weights='imagenet')

img_path = 'dog.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

predictions = model.predict(x)

for item in decode_predictions(predictions, top=3)[0]:
    print(item[1],item[2])

```

NINE LINES OF CODE

is_this_a_dog.py > ...

```python
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
import numpy as np

model = VGG16(weights='imagenet')

img_path = 'dog.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

predictions = model.predict(x)

for item in decode_predictions(predictions, top=3)[0]:
    print(item[1],item[2])
```

NINE LINES OF CODE

is_this_a_dog.py > ...

```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
import numpy as np

model = VGG16(weights='imagenet')

img_path = 'dog.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

predictions = model.predict(x)

for item in decode_predictions(predictions, top=3)[0]:
    print(item[1],item[2])
```

PREDICT


cadec2020 — -bash — 77×10
(tf2) antoncic@NiklasMBP:~/cadec2020$ python is_this_a_dog.py

PREDICT

cadec2020 — -bash — 77×10

```
(tf2) antoncic@NiklasMBP:~/cadec2020$ python is_this_a_dog.py
golden_retriever 0.95973116
Labrador_retriever 0.035258666
kuvasz 0.0008982596
(tf2) antoncic@NiklasMBP:~/cadec2020$
```

PREDICT

cadec2020 — -bash — 77×10

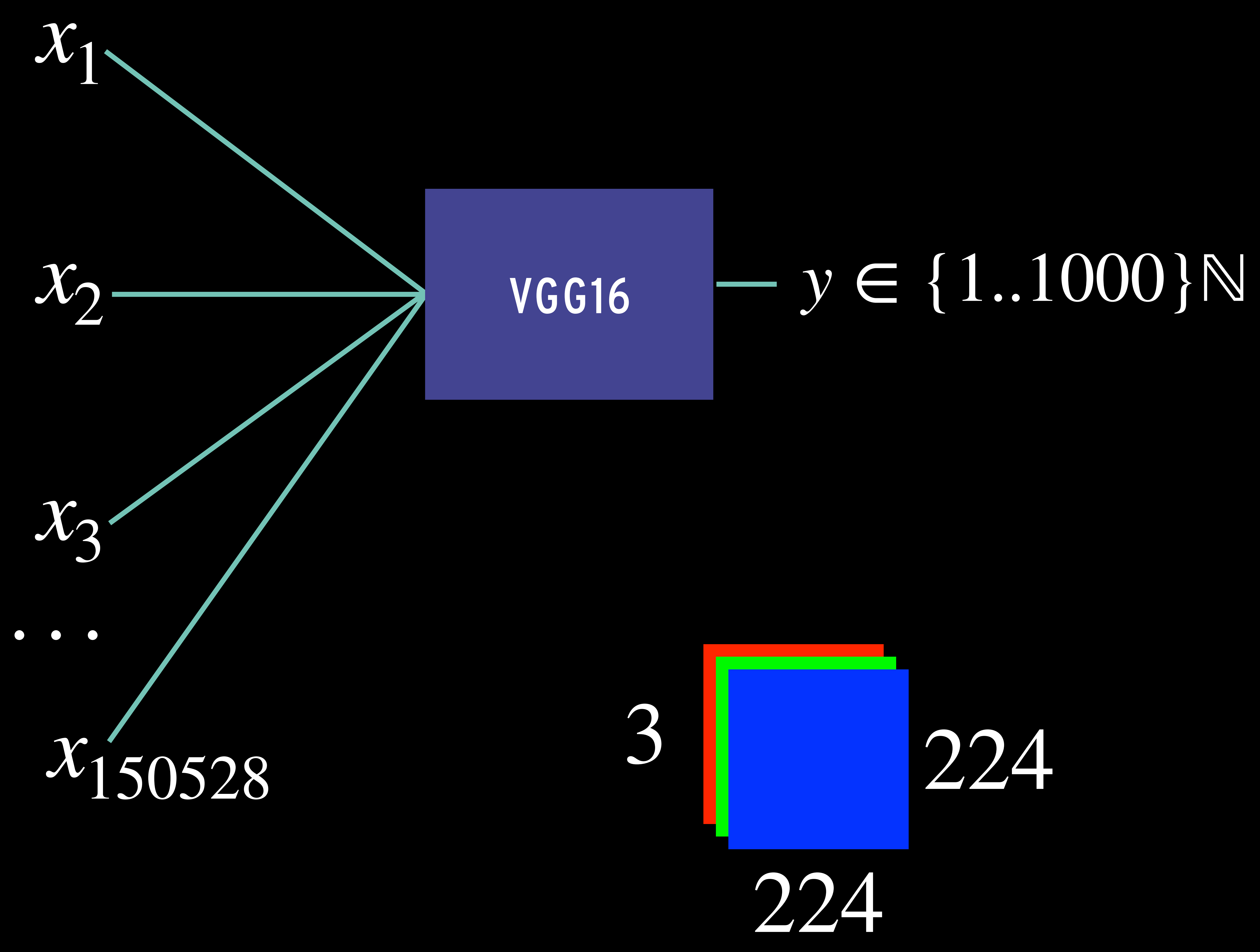
```
(tf2) antoncic@NiklasMBP:~/cadec2020$ python is_this_a_dog.py
golden_retriever 0.95973116
Labrador_retriever 0.035258666
kuvasz 0.0008982596
(tf2) antoncic@NiklasMBP:~/cadec2020$
```

COMPLEX MODELS

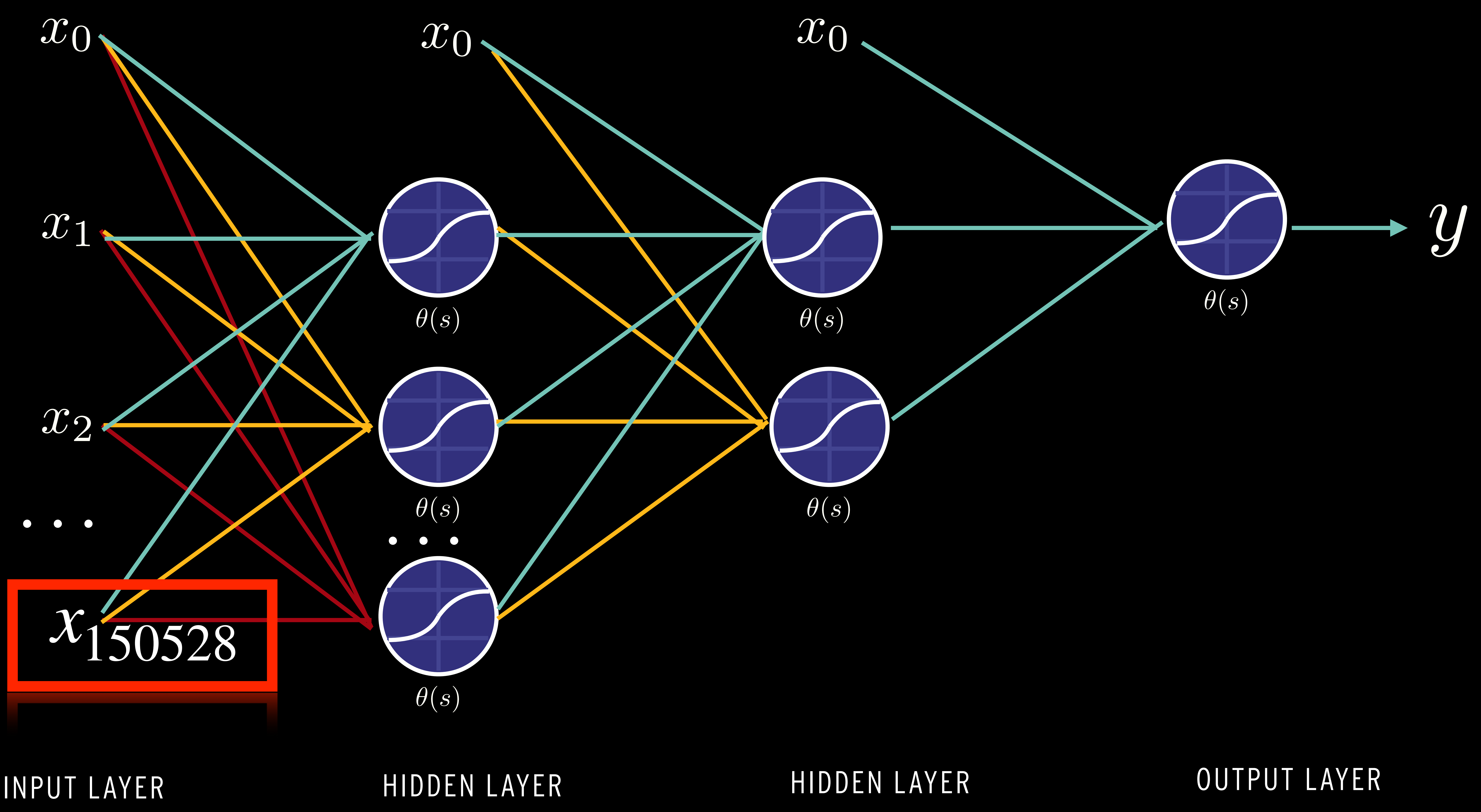
- Simple to classify with a canned deep learning model.
- But why is this a Golden Retriever?

COMPLEX MODELS - VGG16

- VGG16 is a Convolutional Neural Network (CNN)
- 1000 categories
- Inputs: 224x224x3 = 150528
- Trainable weights: 138357544

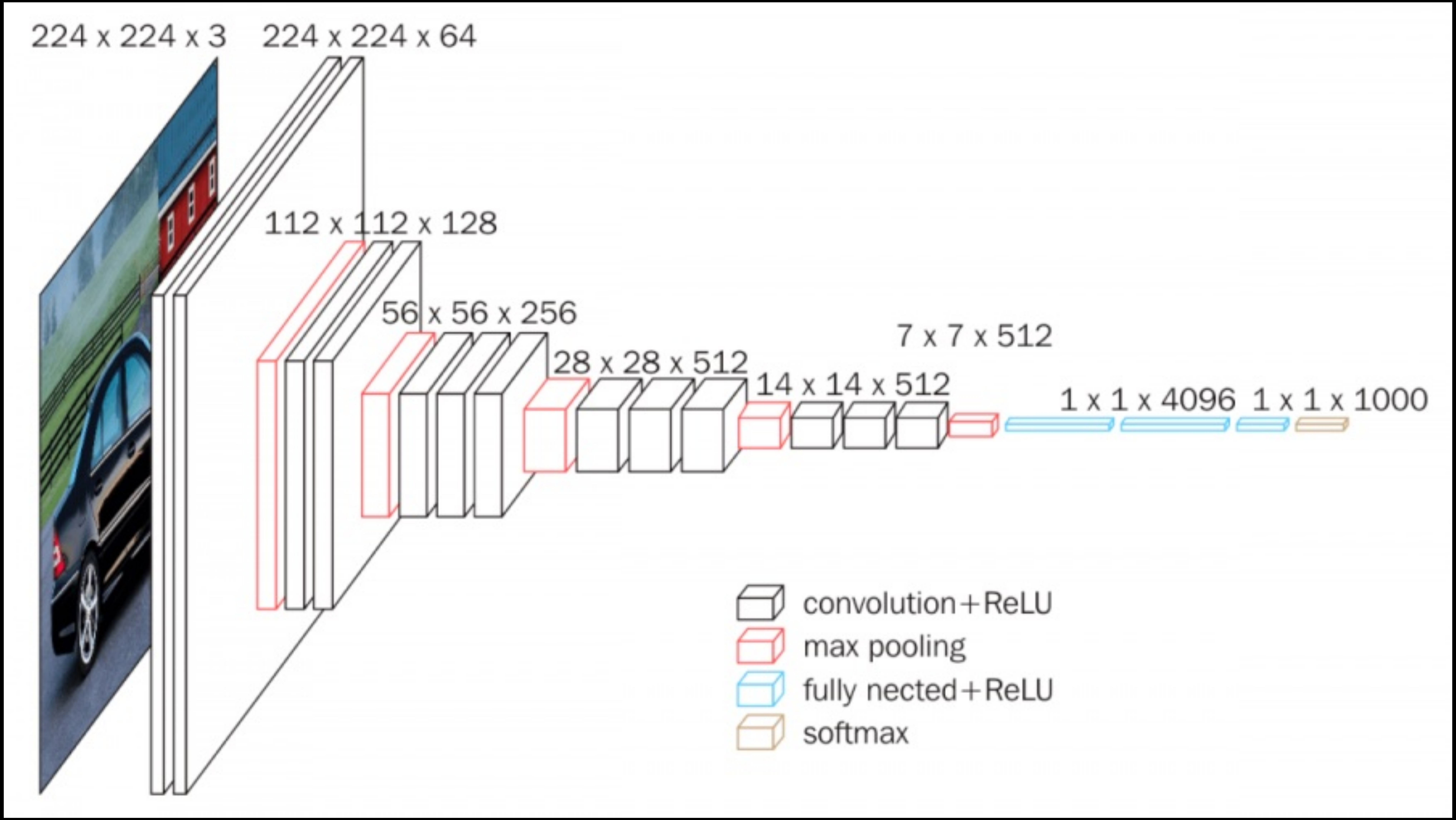


COMPLEX MODELS - CONCEPTUAL NEURAL NETWORK



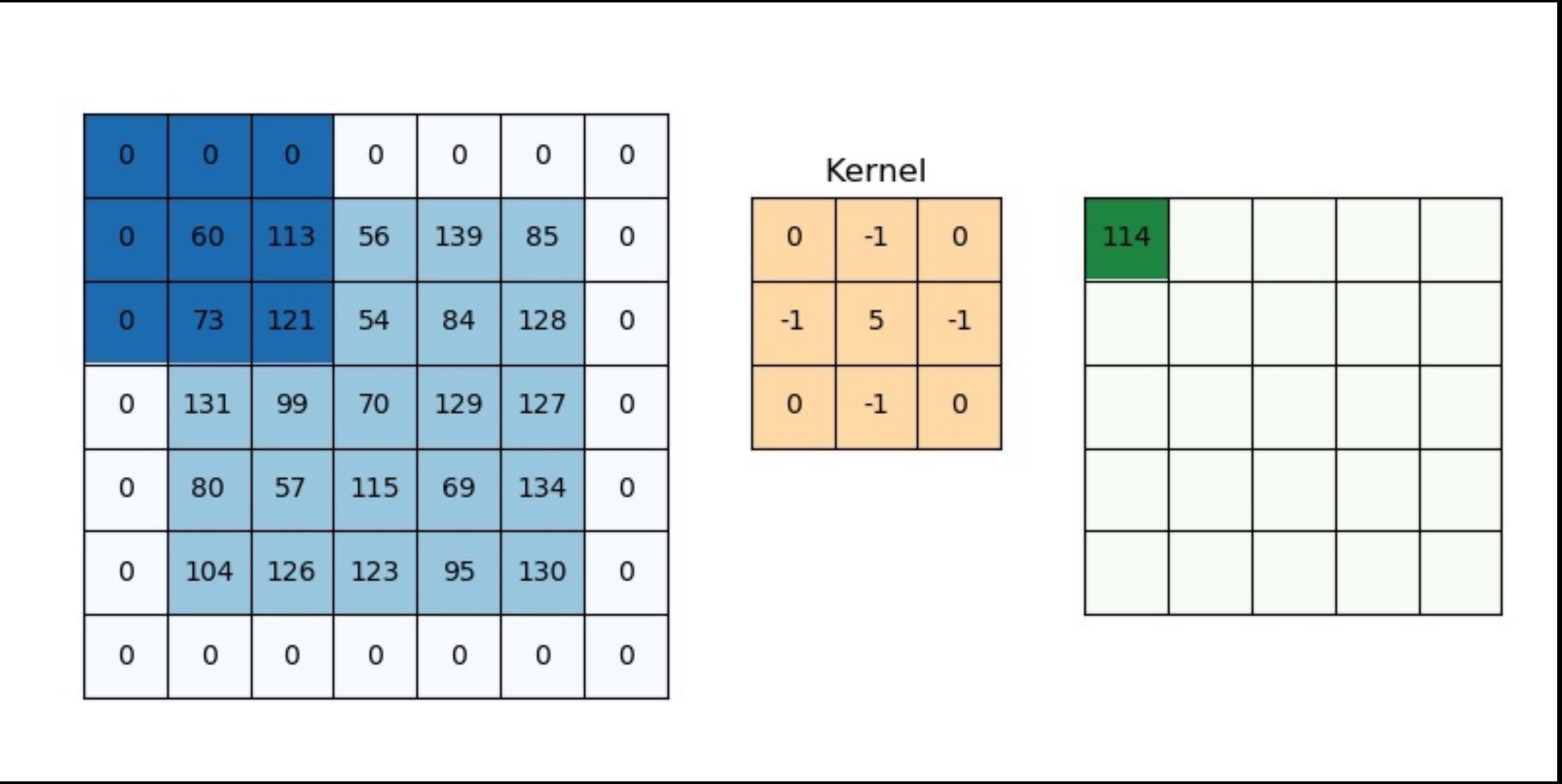
CALLISTA

COMPLEX MODELS - VGG16

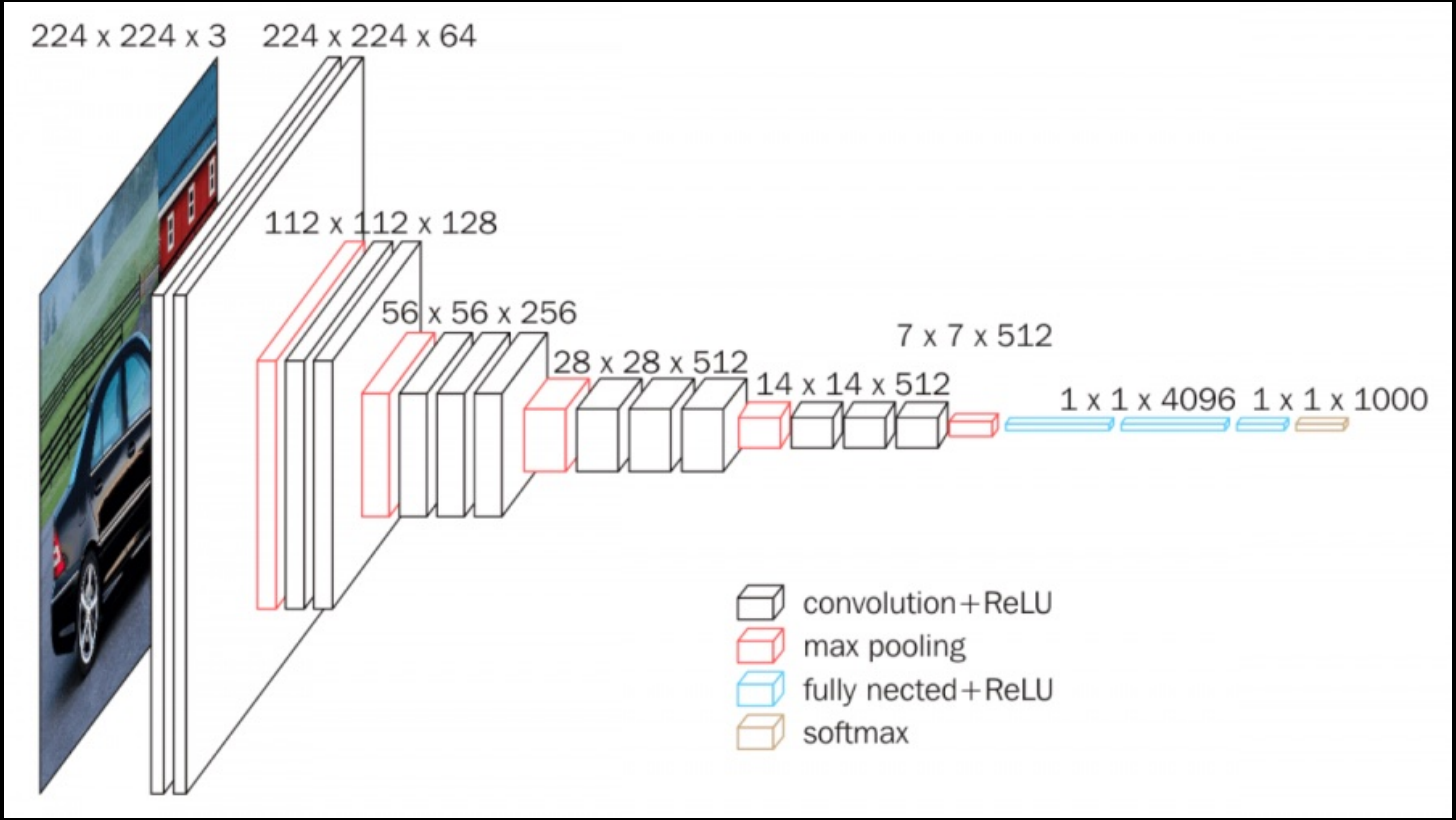


COMPLEX MODELS - CONVOLUTIONS

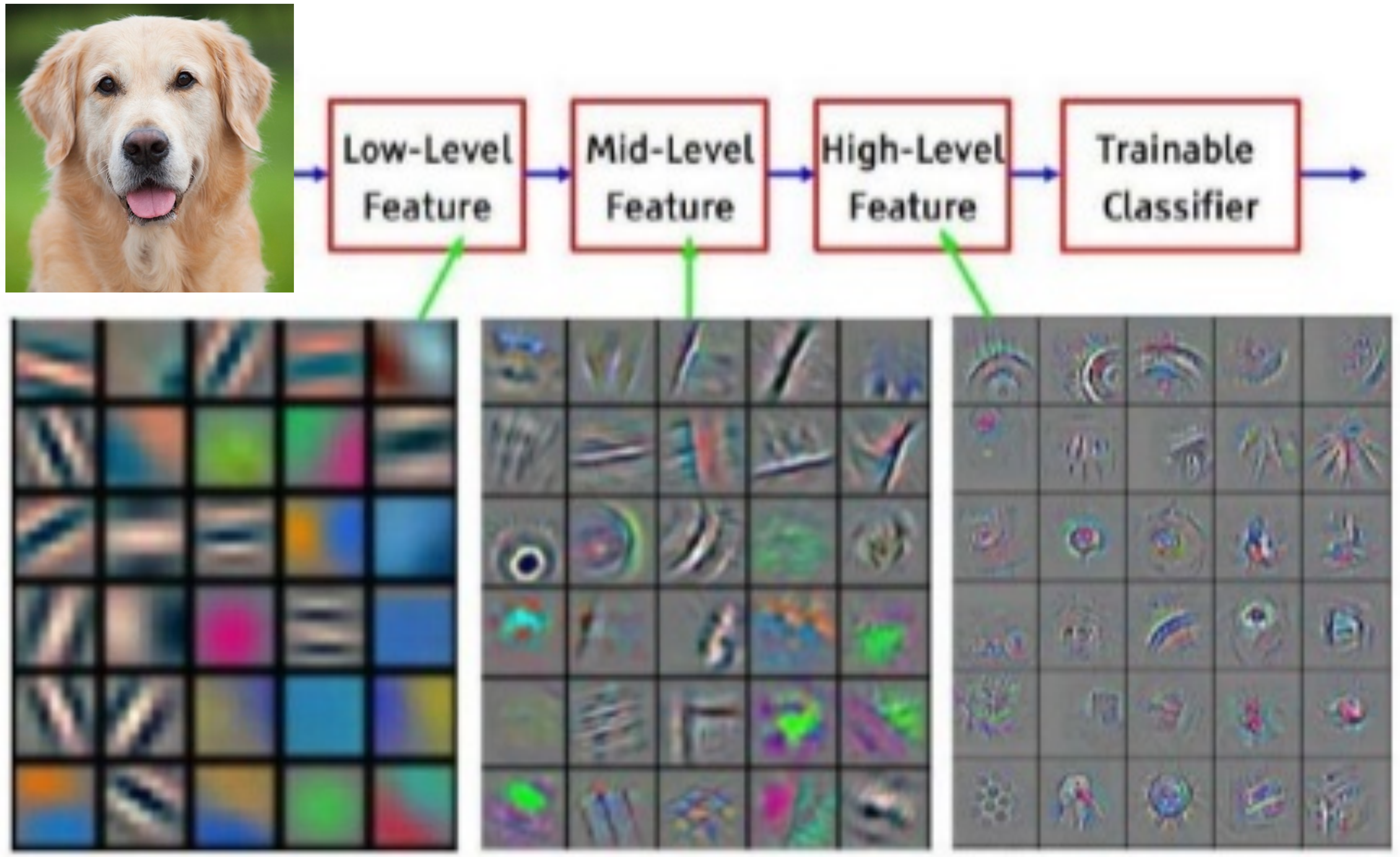
- Applying a learnable filter (or kernel) on a part of the image and then moving it around generating a map of the effect of the filter.



COMPLEX MODELS - VGG16



FEATURE VISUALISATION



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

EXPLANATION MODELS

- The best explanation model is the model itself …
- … but in the complex case that is not possible and thus we need a separate explanation model

?

EXPLANATION MODELS

- The best explanation model is the model itself …
- … but in the complex case that is not possible and thus we need a separate explanation model
- Additive Feature Attribution

!

ADDITIVE FEATURE ATTRIBUTION

- Local explainability

- Local explainability
- Local explanation model

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i$$

ADDITIVE FEATURE ATTRIBUTION

- Local explainability
- Local explanation model
- Simplified input

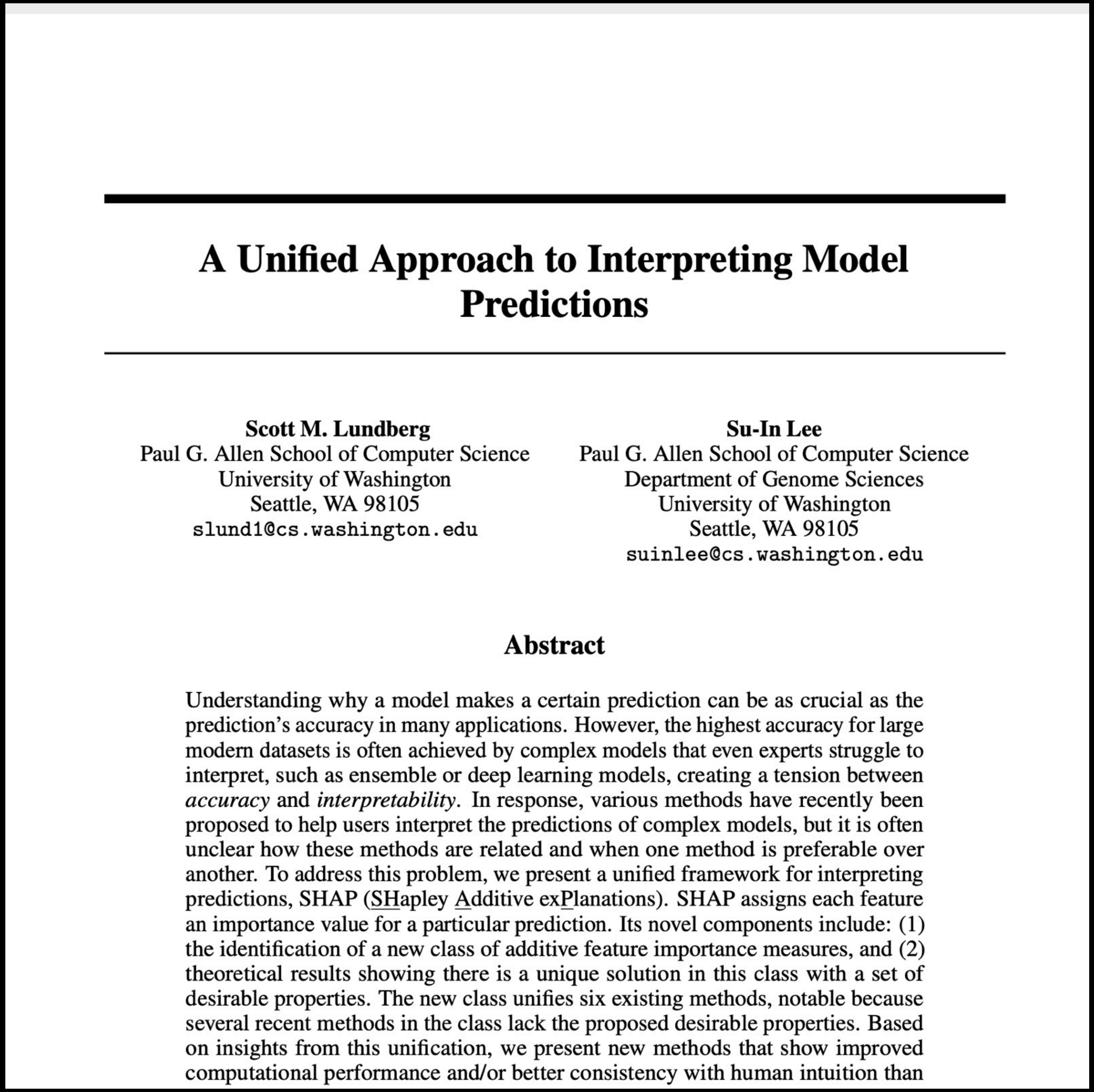
ADDITIVE FEATURE ATTRIBUTION

- Local explainability
- Local explanation model
- Simplified input
- Shapley Values

SHAPLEY VALUES

SHAP - SHAPELY ADDITIVE EXPLANATIONS

- Method proposed by Scott Lundberg & Su-In Lee in 2017
- Theoretically correct
- General, works for all models
- Best model so far …



A Unified Approach to Interpreting Model Predictions

Scott M. Lundberg
Paul G. Allen School of Computer Science
University of Washington
Seattle, WA 98105
slund1@cs.washington.edu

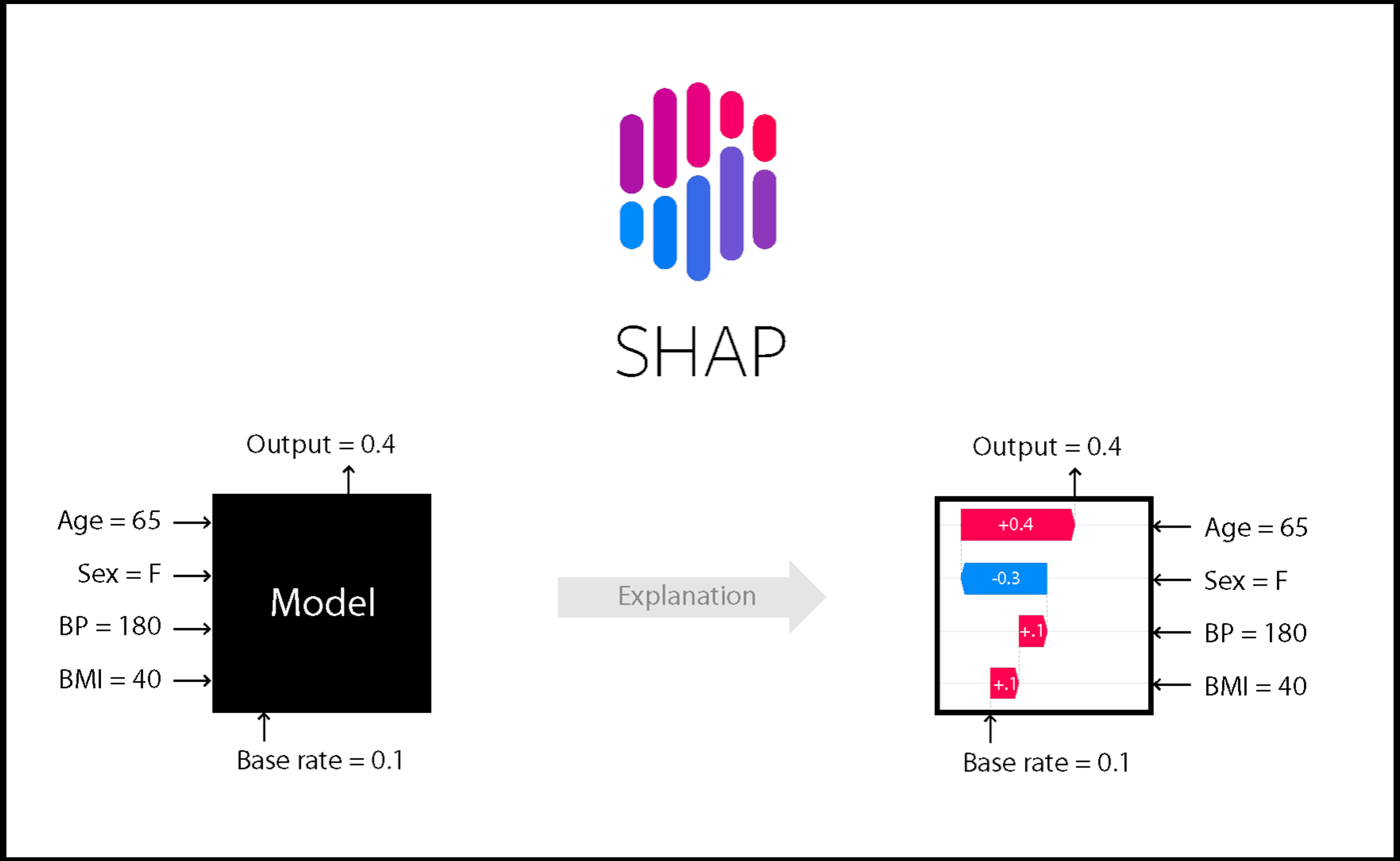
Su-In Lee
Paul G. Allen School of Computer Science
Department of Genome Sciences
University of Washington
Seattle, WA 98105
suinlee@cs.washington.edu

Abstract

Understanding why a model makes a certain prediction can be as crucial as the prediction's accuracy in many applications. However, the highest accuracy for large modern datasets is often achieved by complex models that even experts struggle to interpret, such as ensemble or deep learning models, creating a tension between *accuracy* and *interpretability*. In response, various methods have recently been proposed to help users interpret the predictions of complex models, but it is often unclear how these methods are related and when one method is preferable over another. To address this problem, we present a unified framework for interpreting predictions, SHAP (SHapley Additive exPlanations). SHAP assigns each feature an importance value for a particular prediction. Its novel components include: (1) the identification of a new class of additive feature importance measures, and (2) theoretical results showing there is a unique solution in this class with a set of desirable properties. The new class unifies six existing methods, notable because several recent methods in the class lack the proposed desirable properties. Based on insights from this unification, we present new methods that show improved computational performance and/or better consistency with human intuition than

SHAP

- SHapely Additive exPlanations
- Naive implementation slow, $O(2^m)$
- Python lib from the authors
- Optimized SHAP explainers for different model types



SHAP - GRADIENT EXPLAINER

explain_gradient.py > ...

```python
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
import keras.backend as K
import numpy as np
import json
import shap

# load pre-trained model
model = VGG16(weights='imagenet', include_top=True)

# load 50 random images from ImageNet as background reference
background, dummy = shap.datasets.imagenet50()

# load image to explain
img_path = 'dogs/dog-images/golden.jpg'
img = image.load_img(img_path, target_size=(224, 224))
image_to_explain = image.img_to_array(img)
image_to_explain = np.expand_dims(image_to_explain, axis=0)

# load the ImageNet class names
url = "https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json"
fname = shap.datasets.cache(url)
with open(fname) as f:
    class_names = json.load(f)

```

SHAP - GRADIENT EXPLAINER

explain_gradient.py > ...

```python
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
import keras.backend as K
import numpy as np
import json
import shap

# load pre-trained model
model = VGG16(weights='imagenet', include_top=True)

# load 50 random images from ImageNet as background reference
background, dummy = shap.datasets.imagenet50()

# load image to explain
img_path = 'dogs/dog-images/golden.jpg'
img = image.load_img(img_path, target_size=(224, 224))
image_to_explain = image.img_to_array(img)
image_to_explain = np.expand_dims(image_to_explain, axis=0)

# load the ImageNet class names
url = "https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json"
fname = shap.datasets.cache(url)
with open(fname) as f:
    class_names = json.load(f)
```

SHAP - GRADIENT EXPLAINER

explain_gradient.py > ...

```python
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
import keras.backend as K
import numpy as np
import json
import shap

# load pre-trained model
model = VGG16(weights='imagenet', include_top=True)

# load 50 random images from ImageNet as background reference
background, dummy = shap.datasets.imagenet50()

# load image to explain
img_path = 'dogs/dog-images/golden.jpg'
img = image.load_img(img_path, target_size=(224, 224))
image_to_explain = image.img_to_array(img)
image_to_explain = np.expand_dims(image_to_explain, axis=0)

# load the ImageNet class names
url = "https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json"
fname = shap.datasets.cache(url)
with open(fname) as f:
    class_names = json.load(f)

```

SHAP - GRADIENT EXPLAINER

explain_gradient.py > ...

```python
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
import keras.backend as K
import numpy as np
import json
import shap

# load pre-trained model
model = VGG16(weights='imagenet', include_top=True)

# load 50 random images from ImageNet as background reference
background, dummy = shap.datasets.imagenet50()

# load image to explain
img_path = 'dogs/dog-images/golden.jpg'
img = image.load_img(img_path, target_size=(224, 224))
image_to_explain = image.img_to_array(img)
image_to_explain = np.expand_dims(image_to_explain, axis=0)

# load the ImageNet class names
url = "https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json"
fname = shap.datasets.cache(url)
with open(fname) as f:
    class_names = json.load(f)

```

SHAP - GRADIENT EXPLAINER

explain_gradient.py > ...

```python
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
import keras.backend as K
import numpy as np
import json
import shap

# load pre-trained model
model = VGG16(weights='imagenet', include_top=True)

# load 50 random images from ImageNet as background reference
background, dummy = shap.datasets.imagenet50()

# load image to explain
img_path = 'dogs/dog-images/golden.jpg'
img = image.load_img(img_path, target_size=(224, 224))
image_to_explain = image.img_to_array(img)
image_to_explain = np.expand_dims(image_to_explain, axis=0)

# load the ImageNet class names
url = "https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json"
fname = shap.datasets.cache(url)
with open(fname) as f:
    class_names = json.load(f)

```

SHAP - GRADIENT EXPLAINER

explain_gradient.py > ...

```python
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
import keras.backend as K
import numpy as np
import json
import shap

# load pre-trained model
model = VGG16(weights='imagenet', include_top=True)

# load 50 random images from ImageNet as background reference
background, dummy = shap.datasets.imagenet50()

# load image to explain
img_path = 'dogs/dog-images/golden.jpg'
img = image.load_img(img_path, target_size=(224, 224))
image_to_explain = image.img_to_array(img)
image_to_explain = np.expand_dims(image_to_explain, axis=0)

# load the ImageNet class names
url = "https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json"
fname = shap.datasets.cache(url)
with open(fname) as f:
    class_names = json.load(f)

```

SHAP - GRADIENT EXPLAINER

explain_gradient.py > ...

```python
# load the ImageNet class names
url = "https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json"
fname = shap.datasets.cache(url)
with open(fname) as f:
    class_names = json.load(f)

# explain how the input to a layer of the model explains the top n classes
layer_to_explain = 11
def map2layer(x, layer):
    feed_dict = dict(zip([model.layers[0].input], [preprocess_input(x.copy())]))
    return K.get_session().run(model.layers[layer].input, feed_dict)
e = shap.GradientExplainer(
    (model.layers[layer_to_explain].input, model.layers[-1].output),
    map2layer(background, layer_to_explain),
    local_smoothing=0 # std dev of smoothing noise
)
shap_values,indexes = e.shap_values(map2layer(image_to_explain, layer_to_explain), ranked_outputs=3)

# get the names for the classes
index_names = np.vectorize(lambda x: class_names[str(x)][1])(indexes)

# plot the explanations
shap.image_plot(shap_values, image_to_explain, index_names)
```

SHAP - GRADIENT EXPLAINER

explain_gradient.py > ...

```python
# load the ImageNet class names
url = "https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json"
fname = shap.datasets.cache(url)
with open(fname) as f:
    class_names = json.load(f)

# explain how the input to a layer of the model explains the top n classes
layer_to_explain = 11
def map2layer(x, layer):
    feed_dict = dict(zip([model.layers[0].input], [preprocess_input(x.copy())]))
    return K.get_session().run(model.layers[layer].input, feed_dict)
e = shap.GradientExplainer(
    (model.layers[layer_to_explain].input, model.layers[-1].output),
    map2layer(background, layer_to_explain),
    local_smoothing=0 # std dev of smoothing noise
)
shap_values,indexes = e.shap_values(map2layer(image_to_explain, layer_to_explain), ranked_outputs=3)

# get the names for the classes
index_names = np.vectorize(lambda x: class_names[str(x)][1])(indexes)

# plot the explanations
shap.image_plot(shap_values, image_to_explain, index_names)
```

SHAP - GRADIENT EXPLAINER

explain_gradient.py > ...

```python
# load the ImageNet class names
url = "https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json"
fname = shap.datasets.cache(url)
with open(fname) as f:
    class_names = json.load(f)

# explain how the input to a layer of the model explains the top n classes
layer_to_explain = 11
def map2layer(x, layer):
    feed_dict = dict(zip([model.layers[0].input], [preprocess_input(x.copy())]))
    return K.get_session().run(model.layers[layer].input, feed_dict)
e = shap.GradientExplainer(
    (model.layers[layer_to_explain].input, model.layers[-1].output),
    map2layer(background, layer_to_explain),
    local_smoothing=0 # std dev of smoothing noise
)
shap_values,indexes = e.shap_values(map2layer(image_to_explain, layer_to_explain), ranked_outputs=3)

# get the names for the classes
index_names = np.vectorize(lambda x: class_names[str(x)][1])(indexes)

# plot the explanations
shap.image_plot(shap_values, image_to_explain, index_names)
```

SHAP - GRADIENT EXPLAINER

explain_gradient.py > ...

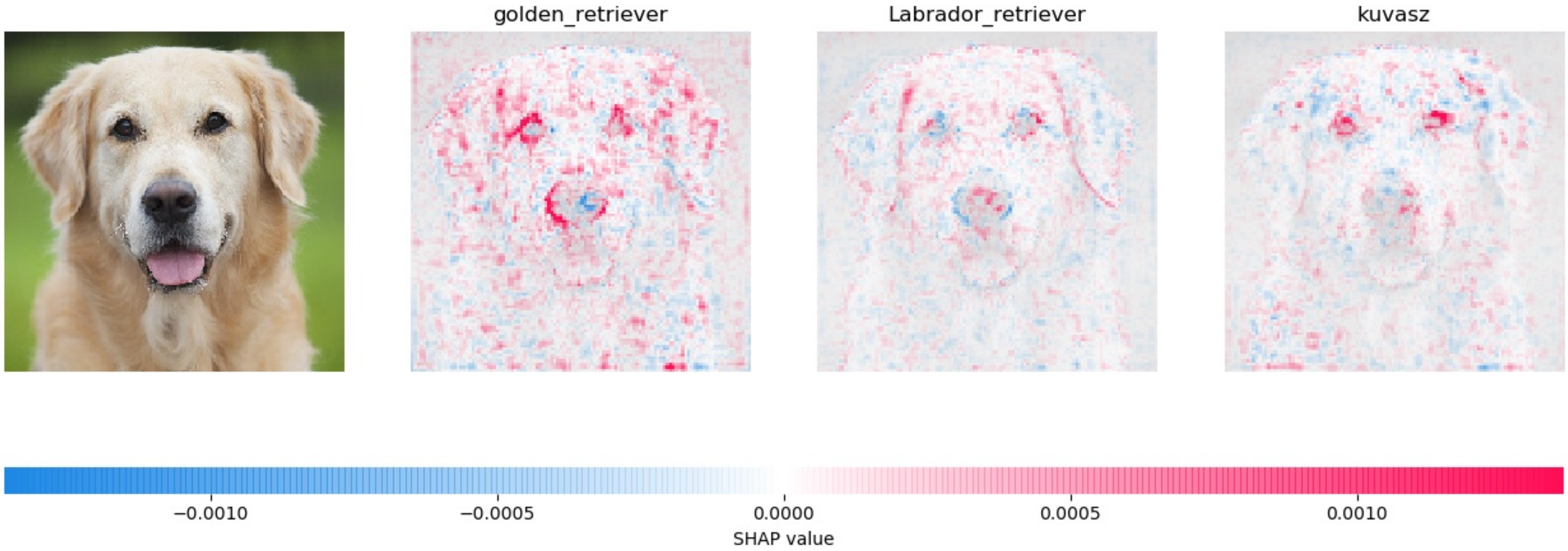
```python
# load the ImageNet class names
url = "https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json"
fname = shap.datasets.cache(url)
with open(fname) as f:
    class_names = json.load(f)

# explain how the input to a layer of the model explains the top n classes
layer_to_explain = 11
def map2layer(x, layer):
    feed_dict = dict(zip([model.layers[0].input], [preprocess_input(x.copy())]))
    return K.get_session().run(model.layers[layer].input, feed_dict)
e = shap.GradientExplainer(
    (model.layers[layer_to_explain].input, model.layers[-1].output),
    map2layer(background, layer_to_explain),
    local_smoothing=0 # std dev of smoothing noise
)
shap_values,indexes = e.shap_values(map2layer(image_to_explain, layer_to_explain), ranked_outputs=3)

# get the names for the classes
index_names = np.vectorize(lambda x: class_names[str(x)][1])(indexes)

# plot the explanations
shap.image_plot(shap_values, image_to_explain, index_names)
```

SHAP - GRADIENT EXPLAINER

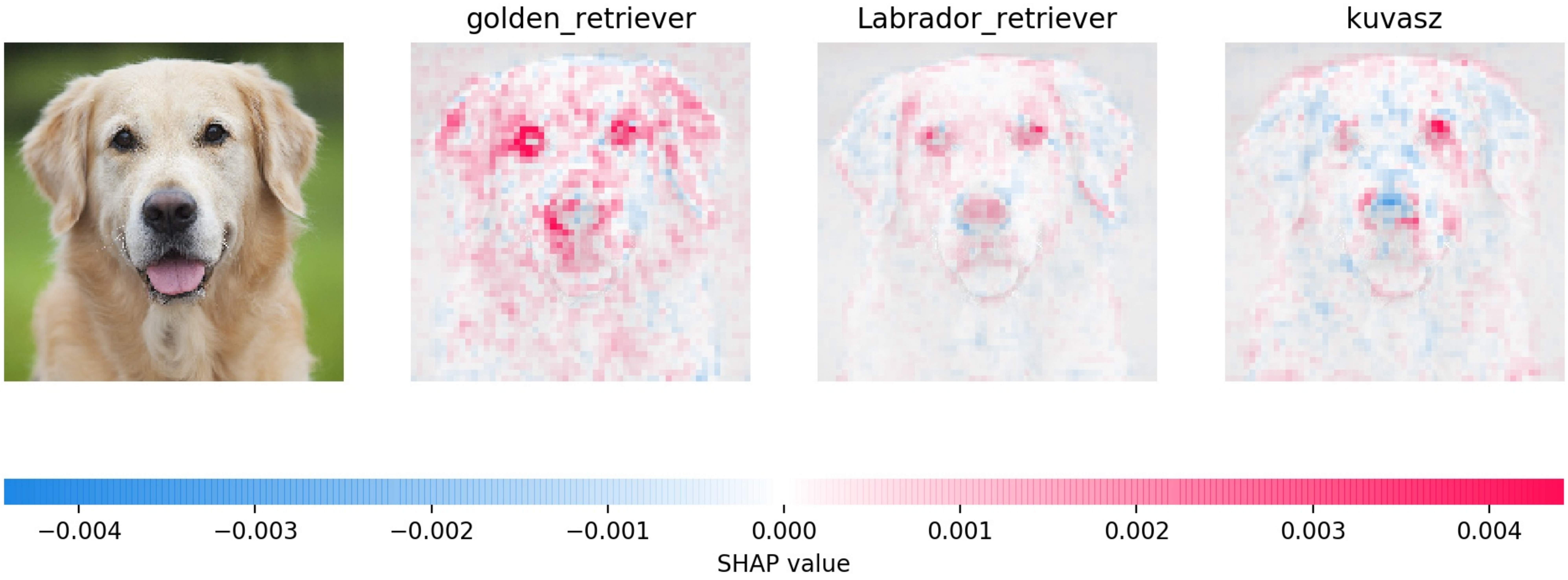
explain_gradient.py > ...

```python
# load the ImageNet class names
url = "https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json"
fname = shap.datasets.cache(url)
with open(fname) as f:
    class_names = json.load(f)

# explain how the input to a layer of the model explains the top n classes
layer_to_explain = 11
def map2layer(x, layer):
    feed_dict = dict(zip([model.layers[0].input], [preprocess_input(x.copy())]))
    return K.get_session().run(model.layers[layer].input, feed_dict)
e = shap.GradientExplainer(
    (model.layers[layer_to_explain].input, model.layers[-1].output),
    map2layer(background, layer_to_explain),
    local_smoothing=0 # std dev of smoothing noise
)
shap_values,indexes = e.shap_values(map2layer(image_to_explain, layer_to_explain), ranked_outputs=3)

# get the names for the classes
index_names = np.vectorize(lambda x: class_names[str(x)][1])(indexes)

# plot the explanations
shap.image_plot(shap_values, image_to_explain, index_names)
```

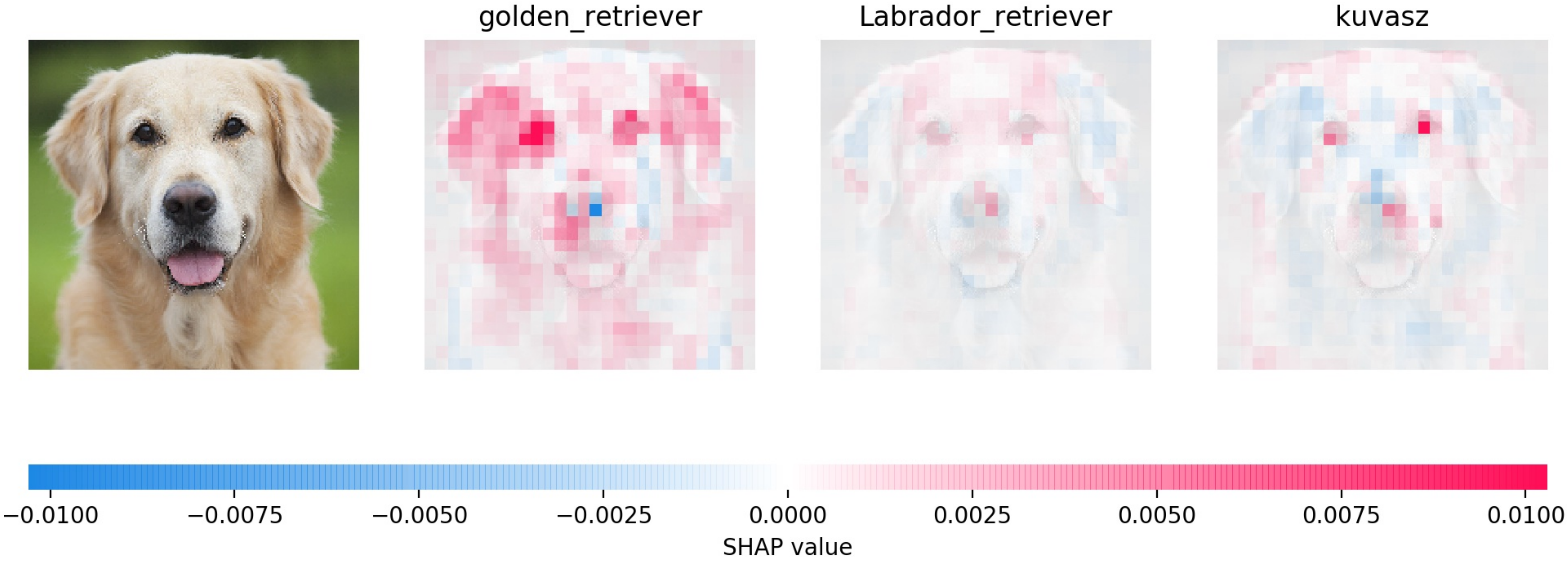
SHAP EXPLAINATION



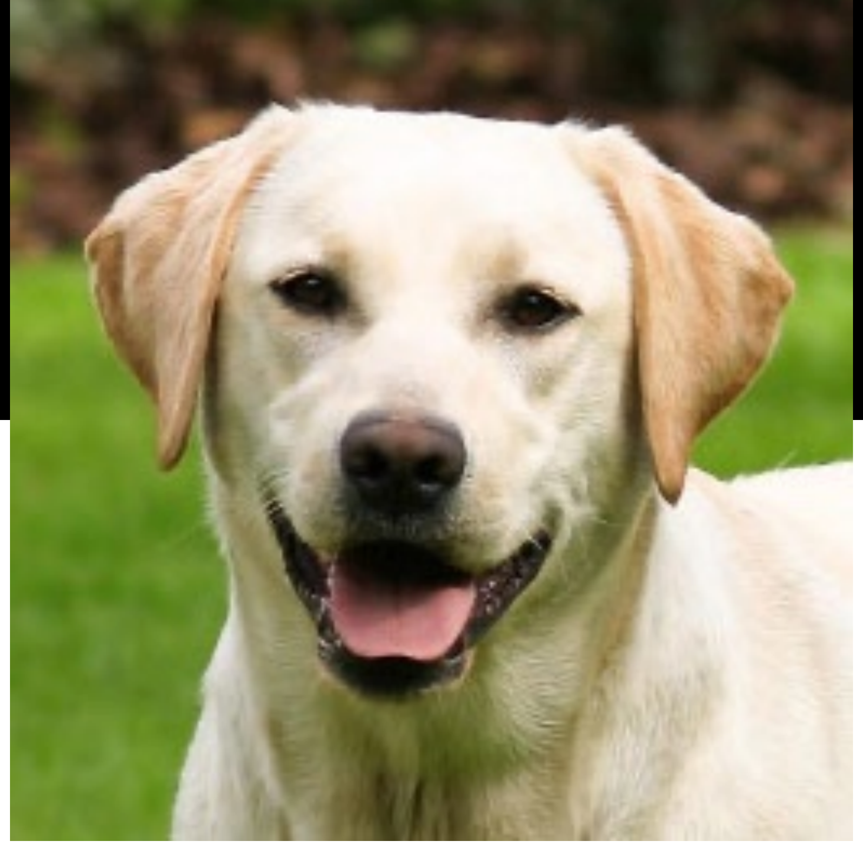
SHAP EXPLAINATION

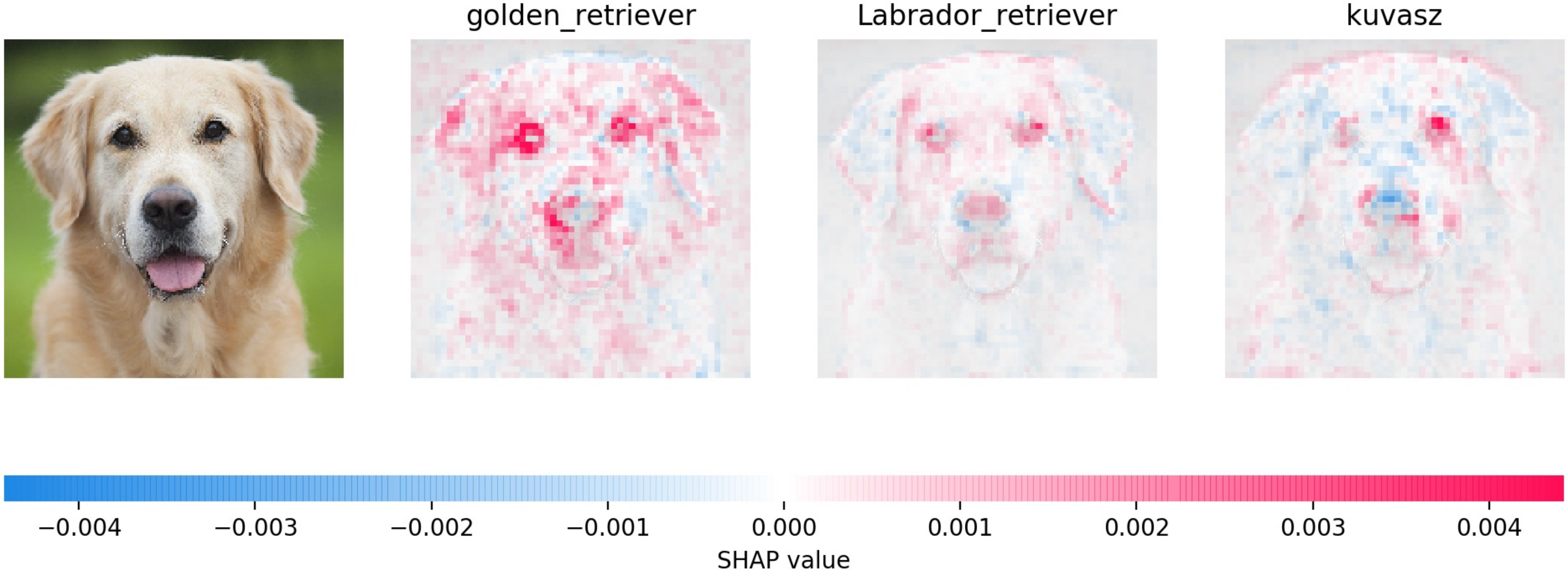


SHAP EXPLAINATION



SHAP EXPLAINATION





THE SO CALLED "SLUTKLÄMM"

- Complex models need explanations for several reasons

- The field of Interpretability and Explainability is still evolving

- SHAP seems to be the best method right now, try it!

CRED

- Marko Cotra, How well do you know your model?
  - GAIA presentation: https://www.youtube.com/watch?v=-PkDvqkyNVI
- SHAP
  - Code: https://github.com/slundberg/shap
  - Paper: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf
- Article in SvD:
  - https://www.svd.se/obehorig-algoritm-tar-beslut-i-socialtjansten
- TensorFlow and Keras
  - Docs: https://www.tensorflow.org/
- GAIA 2020
  - Conference: https://www.gaia.fish/conf

REKLAM

Events Get Involved About Contact The Conference

The 2020 Conference

GAIA organises a one-day conference for people with an interest in artificial intelligence and data science with the focus on what is going on within the field in Gothenburg.

The aim is to create an environment for learning, networking, and knowledge-sharing among individuals, companies, organisations, and academia with a common interest. The conference focuses on applied machine learning and data science and introduces talks of diverse content given by enthusiastic people from the field, many with local connections.

Our last conference attracted over 550 people and was sold out. The third GAIA conference will take place on **April 29th 2020** at Svenska Mässan!

Ticket sales have not yet opened