BST 261: Data Science II
Lecture 6
Convolutional Neural Networks (CNNs): Data Augmentation

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

Recipe of the Day!

Iced vanilla and caramel profiteroles

Paper Presentations

You Only Look Once: Unified, Real-Time Object Detection (2016)

JOSEPH REDMON, SANTOSH DIVVALAY, ROSS GIRSHICK{, ALI FARHADIY

UNIVERSITY OF WASHINGTON, ALLEN INSTITUTE FOR AI, FACEBOOK AI RESEARCH

PRESENTED BY EVAN GOLDBERG

Introduction

- A new approach to object detection
- Framed as a regression problem to determine bounding boxes and class probabilities
- Incredibly fast (45 FPS – 150 FPS)
- Makes less background errors in exchange for more localization errors and lower accuracy
- Outperforms top detection methods on natural images and artwork

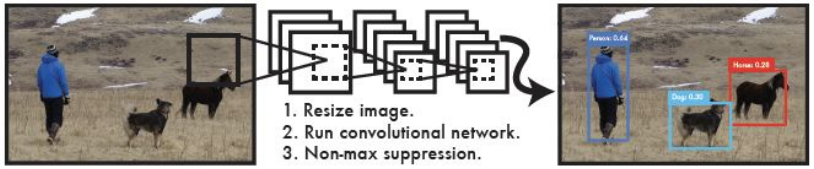


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

Unified Detection

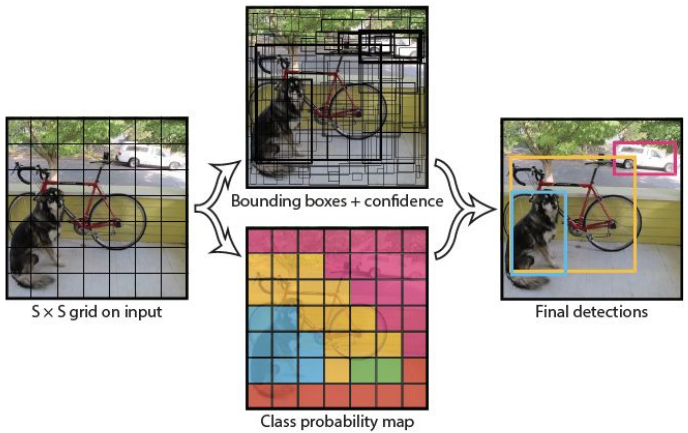


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

- Unify separate components of object detection into a single neural network
- 24 convolutional layers followed by 2 fully connected layers
- Each layer has a Leaky ReLU activation function, except for the last layer which uses a linear activation function
- Optimized for sum-squared error in the output
- Predicts multiple bounding boxes per grid cell
- Can be optimized end-to-end

The Model

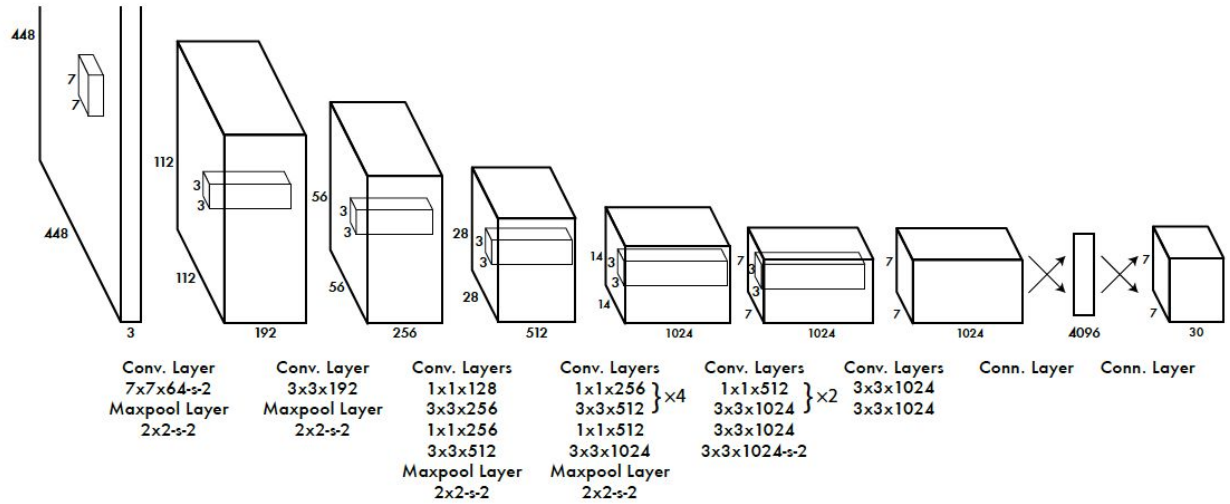


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

Comparisons to Other Systems

- Deformable Parts Models (DPM) are slower and less accurate
  - Use a sliding window approach to object detection
- R-CNNs have some similarities to YOLO
  - Each proposes potential bounding boxes and scores those boxes using convolutional features
  - YOLO proposes far fewer boxes
  - R-CNNs are also slower
- Deep MultiBox cannot perform object detection
- OverFeat optimizes for localization, not detection performance
- MultiGrasp is similar to YOLO but doesn't identify what objects are

Experiments

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Table 1: Real-Time Systems on PASCAL VOC 2007. Comparing the performance and speed of fast detectors. Fast YOLO is the fastest detector on record for PASCAL VOC detection and is still twice as accurate as any other real-time detector. YOLO is 10 mAP more accurate than the fast version while still well above real-time in speed.

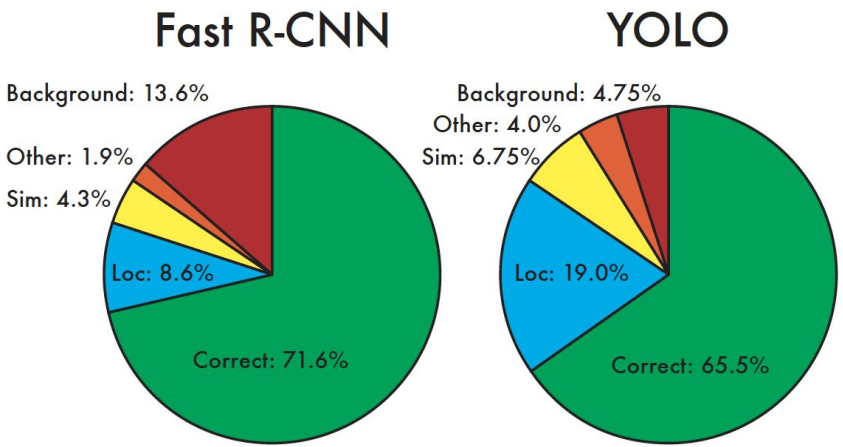


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

Experiments

VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR_CNN_MORE_DATA [11]	**73.9**	**85.5**	**82.9**	**76.6**	**57.8**	**62.7**	**79.4**	77.2	86.6	**55.0**	**79.1**	**62.2**	87.0	**83.4**	**84.7**	78.9	45.3	73.4	65.8	80.3	74.0
HyperNet_VGG	71.4	84.2	78.5	73.6	55.6	53.7	78.7	**79.8**	87.7	49.6	74.9	52.1	86.0	81.7	83.3	**81.8**	**48.6**	**73.5**	59.4	79.9	65.7
HyperNet_SP	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6
Fast R-CNN + YOLO	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	**89.4**	49.4	75.5	57.0	**87.5**	80.9	81.0	74.7	41.8	71.5	68.5	**82.1**	67.2
MR_CNN_S_CNN [11]	70.7	85.0	79.6	71.5	55.3	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1
Faster R-CNN [28]	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
DEEP_ENS_COCO	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	**68.8**	75.9	71.4
NoC [29]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1
Fast R-CNN [14]	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	**87.5**	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
UMICH_FGS_STRUCT	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2
NUS_NIN_C2000 [7]	63.8	80.2	73.8	61.9	43.7	43.0	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	38.6	68.3	58.0	68.7	63.3
BabyLearning [7]	63.2	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6
NUS_NIN	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7
R-CNN VGG BB [13]	62.4	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3
R-CNN VGG [13]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7
YOLO	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
Feature Edit [33]	56.3	74.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8
R-CNN BB [13]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1
SDS [16]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7
R-CNN [13]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6

Table 3: PASCAL VOC 2012 Leaderboard. YOLO compared with the full `comp4` (outside data allowed) public leaderboard as of November 6th, 2015. Mean average precision and per-class average precision are shown for a variety of detection methods. YOLO is the only real-time detector. Fast R-CNN + YOLO is the forth highest scoring method, with a 2.3% boost over Fast R-CNN.

Real Time Detection in the Wild

- YOLO is ideal for computer vision applications
- Functions like a tracking system, detecting objects as they move around and change appearance
- YOLO is also very good at recgonizing natural landscapes and identifying objects in art

Real Time Detection in the Wild

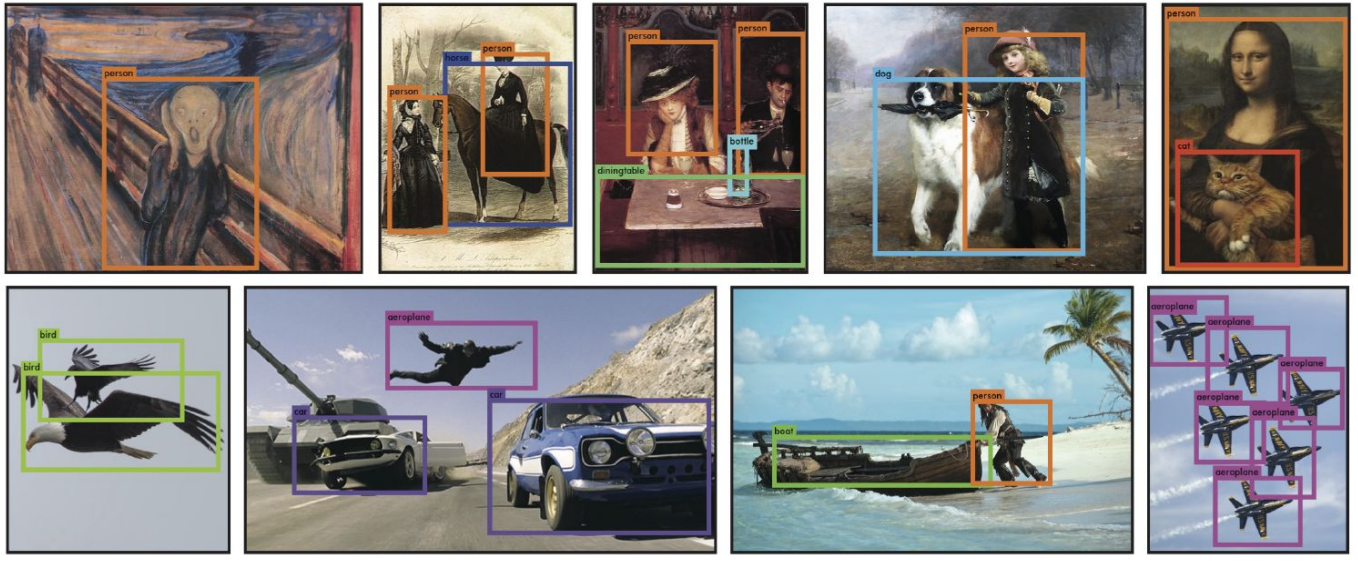


Figure 6: Qualitative Results. YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane.

Limitations

- Imposes strong spatial constraints on bounding boxes which limits the number of nearby objects that the model can predict
  - Struggles with groups such as a flock of birds
- Struggles to generalize to objects in new or unusual aspect ratios due to learning to predict bounding boxes from data
- Loss function treats errors the same in small bounding boxes and in large bounding boxes
  - Small error in a large box is not a big deal, but it can be a bigger deal in a small box

Prediction of cardiovascular risk factors from retinal fundus photographs via deep learning

Authors: Ryan Poplin et al.

Presenter: Xinyi Li

Background

- Cardiovascular risk calculator: e.g. SCORE system

age + sex + smoking + SBP + total cholesterol **+ retinal image** ~ CVD risk

- Retinal fundus image: quick, cheap, non-invasive

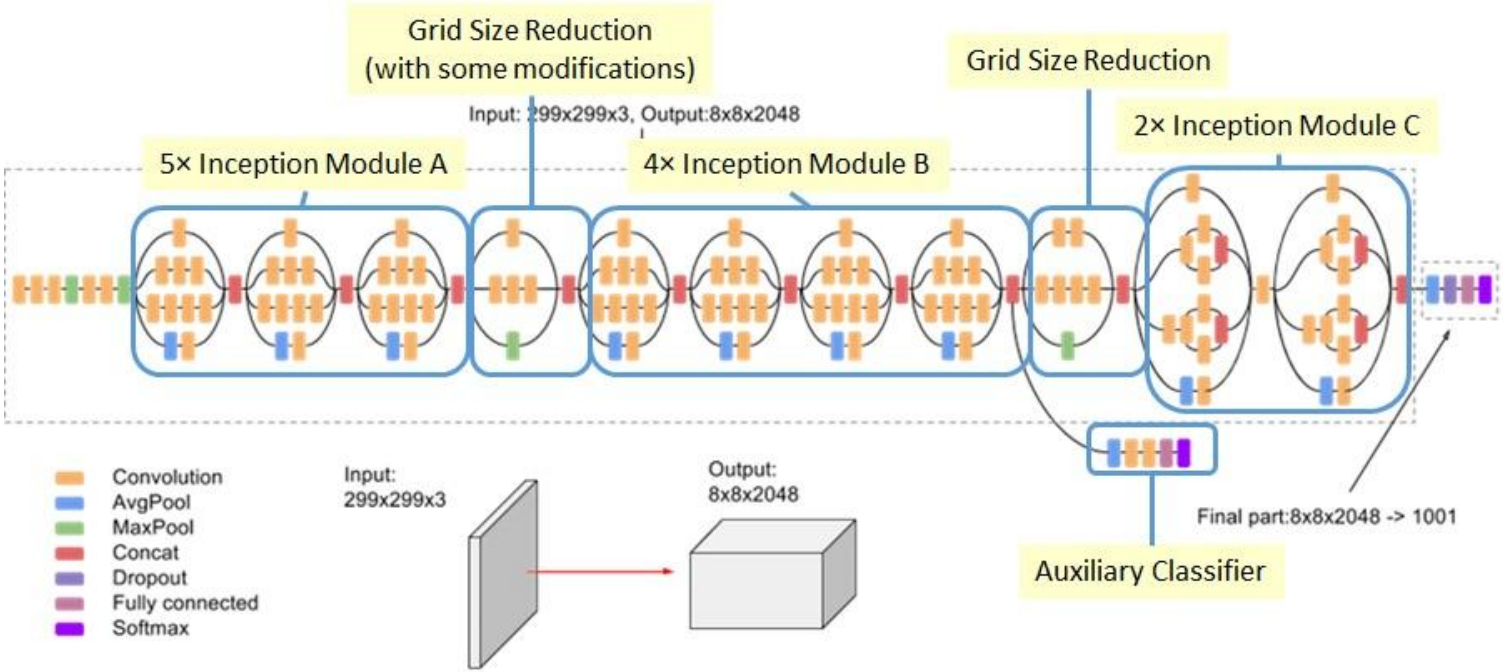
- Markers of CVD (hypertensive retinopathy, cholesterol emboli);
- Features of blood vessels (caliber/diameter, tortuosity, microvascular changes)

Dataset

	Training set		**Validation set**	
	UK Biobank	EyePACS	UK Biobank	EyePACS
Number of patients	48,101	236,234	12,026	999
Number of images	96,082	1,682,938	24,008	1,958

Method

- Google’s Inception v3 CNN



Result – risk factors

Table 2 | Algorithm performance on predicting cardiovascular risk factors in the two validation sets

Predicted risk factor (evaluation metric)	UK Biobank validation dataset ($n = 12{,}026$ patients)		EyePACS-2K validation dataset ($n = 999$ patients)	
	Algorithm (95% CI)	Baseline	Algorithm (95% CI)	Baseline
Age: MAE, years (95% CI)	3.26 (3.22,3.31) <	7.06 (6.98,7.13)	3.42 (3.23,3.61) <	8.48 (8.07,8.90)
Age: R^2 (95% CI)	0.74 (0.73,0.75)	0.00	0.82 (0.79,0.84)	0.00
Gender: AUC (95% CI)	0.97 (0.966,0.971) >	0.50	0.97 (0.96,0.98) >	0.50
Current smoker: AUC (95% CI)	0.71 (0.70,0.73) >	0.50	n/a	n/a
HbA1c: MAE, % (95% CI)	n/a	n/a	1.39 (1.29,1.50) <	1.67 (1.58,1.77)
HbA1c: R^2 (95% CI)	n/a	n/a	0.09 (0.03,0.16)	0.00
SBP: MAE, mmHg (95% CI)	11.35 (11.18,11.51) <	14.57 (14.38,14.77)	n/a	n/a
SBP: R^2 (95% CI)	0.36 (0.35,0.37)	0.00	n/a	n/a
DBP: MAE, mmHg (95% CI)	6.42 (6.33,6.52) <	7.83 (7.73,7.94)	n/a	n/a
DBP: R^2 (95% CI)	0.32 (0.30,0.33)	0.00	n/a	n/a
BMI: MAE (95% CI)	3.29 (3.24,3.34) <	3.62 (3.57,3.68)	n/a	n/a
BMI: R^2 (95% CI)	0.13 (0.11,0.14)	0.00	n/a	n/a

95% CIs on the metrics were calculated with 2,000 bootstrap samples (Methods). For continuous risk factors (such as age), the baseline value is the MAE of predicting the mean value for all patients.

Table 3 | Model accuracy versus baseline for predicting various continuous risk factors within a given margin

Predicted risk factor	UK Biobank validation dataset (n = 12,026 patients)				EyePACS-2K validation dataset (n = 999 patients)			
	Error margin	Model accuracy (%)	Baseline accuracy (%)[a]	*P* value	Error margin	Model accuracy (%)	Baseline accuracy (%)[a]	*P* value
Age (years)	±1	20	11	<0.0001	±1	20%	13%	<0.0001
	±3	54	29	<0.0001	±3	56%	28%	<0.0001
	±5	78	44	<0.0001	±5	79%	43%	<0.0001
SBP (mmHg)	±5	29	22	<0.0001				
	±10	53	43	<0.0001				
	±15	72	60	<0.0001				
DBP (mmHg)	±3	30	25	<0.0001				
	±5	46	41	<0.0001				
	±10	79	71	<0.0001				
BMI	±1	21	20	0.02				
	±3	57	54	<0.0001				
	±5	80	77	<0.0001				
HbA1c (%)					±0.5	31	35	0.995
					±1	54	54	0.486
					±2	79	78	0.255

Result – 5-year MACE

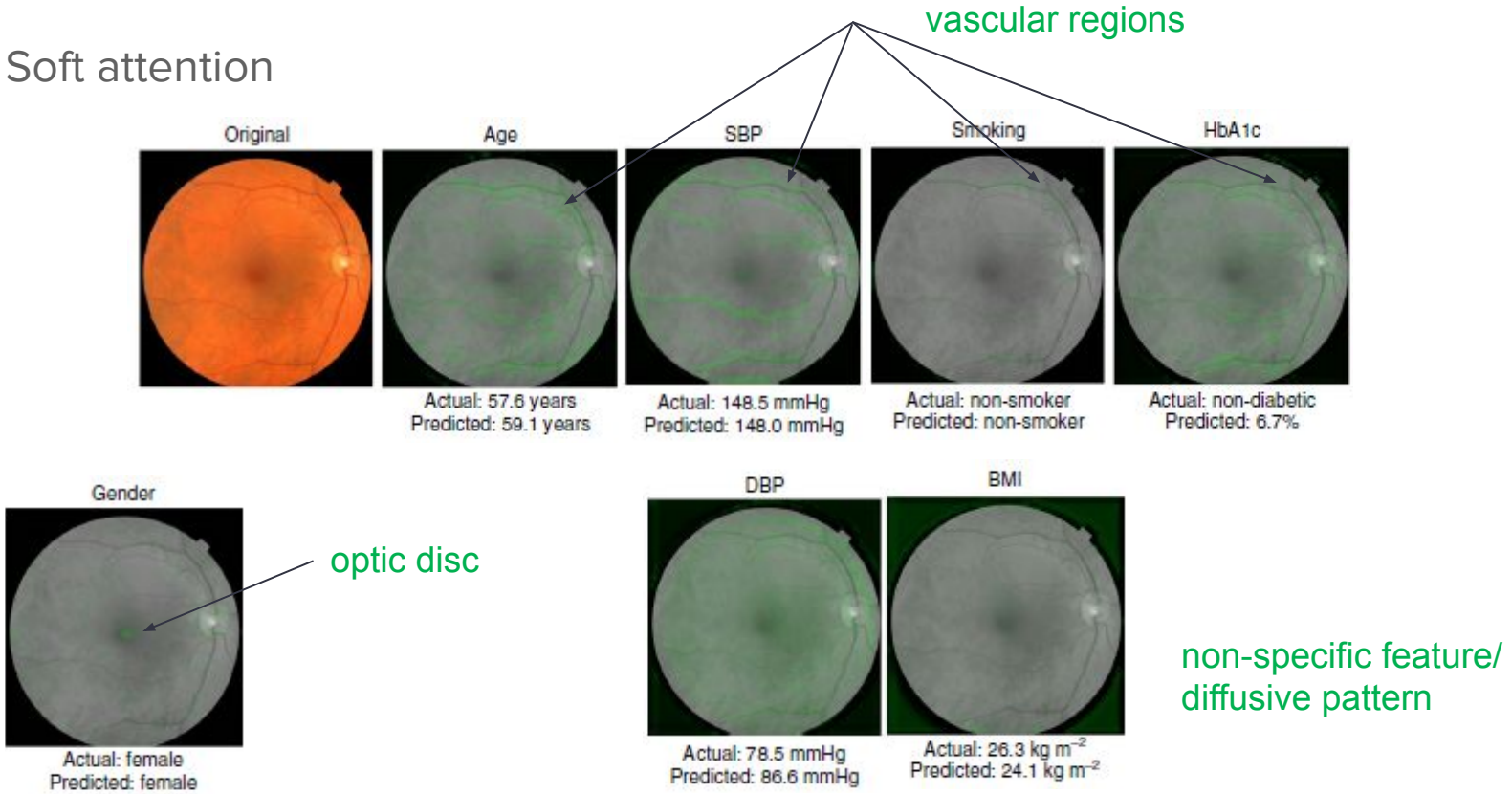
Table 5 | Predicting five-year MACE in the UK Biobank validation dataset using various input variables

Risk factor(s) or model used for the prediction	AUC (95% CI)
Age only	0.66 (0.61,0.71)
SBP only	0.66 (0.61,0.71)
BMI only	0.62 (0.56,0.67)
Gender only	0.57 (0.53,0.62)
Current smoker only	0.55 (0.52,0.59)
Algorithm only	0.70 (0.65,0.74)
Age + SBP + BMI + gender + current smoker	0.72 (0.68,0.76)
Algorithm + age + SBP + BMI + gender + current smoker	0.73 (0.69,0.77)
SCORE[6,7]	0.72 (0.67,0.76)
Algorithm + SCORE	0.72 (0.67,0.76)

- MACE data only available in UK Biobank;
- 481 events in training set;
- 151 in validation set

Attention maps for risk factors prediction

- Soft attention



Limitation

- Small sample size
- Low R-square for some risk factors

HbA1c: MAE, % (95% CI)	n/a	n/a	1.39 (1.29,1.50)	1.67 (1.58,1.77)
HbA1c: R^2 (95% CI)	n/a	n/a	0.09 (0.03,0.16)	0.00
SBP: MAE, mmHg (95% CI)	11.35 (11.18,11.51)	14.57 (14.38,14.77)	n/a	n/a
SBP: R^2 (95% CI)	0.36 (0.35,0.37)	0.00	n/a	n/a
DBP: MAE, mmHg (95% CI)	6.42 (6.33,6.52)	7.83 (7.73,7.94)	n/a	n/a
DBP: R^2 (95% CI)	0.32 (0.30,0.33)	0.00	n/a	n/a
BMI: MAE (95% CI)	3.29 (3.24,3.34)	3.62 (3.57,3.68)	n/a	n/a
BMI: R^2 (95% CI)	0.13 (0.11,0.14)	0.00	n/a	n/a

- Other risk factors information unavailable for predicting MACE (lipid, diabetes)

Summary

- CVD risk factors are present and quantifiable in retinal images
- Improve risk stratification
- Help understanding how CVD and risk factors affect retinal

Cardiologist-Level Arrhythmia Detection with Convolutional Neural Networks

Pranav Rajpurkar, Awni Y. Hannun, Masoumeh Haghpanahi, Codie Bourn, Andrew Y. Ng

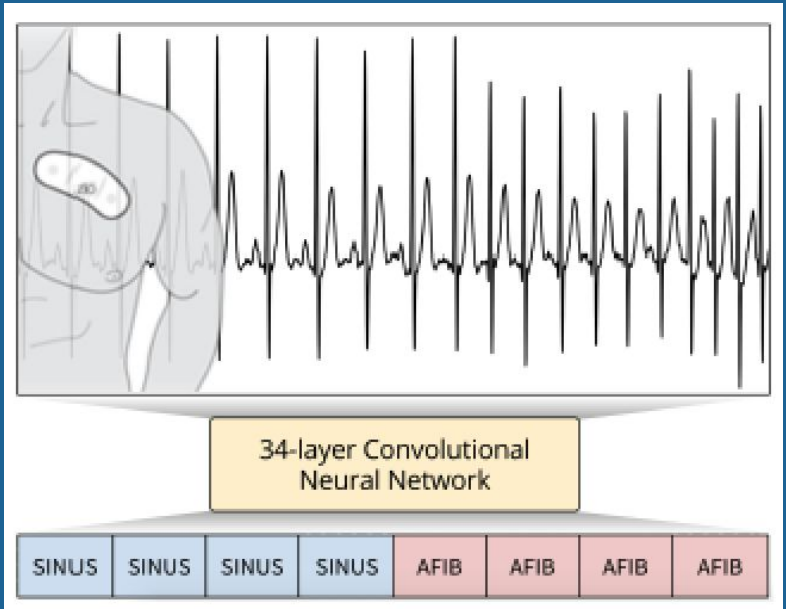
Presentation: Sarah Kalia

April 8, 2020

Background

- Goal: Automated identification and classification of 12 types of arrhythmia (abnormal heart rhythm)
- Recordings of electrical activity of the heartbeat from a single-lead electrocardiogram (ECG)
- Challenges:
  - Recognize distinct wave types and complex relationships between them
  - Variability across patients
  - Noise
  - Previous datasets insufficiently large

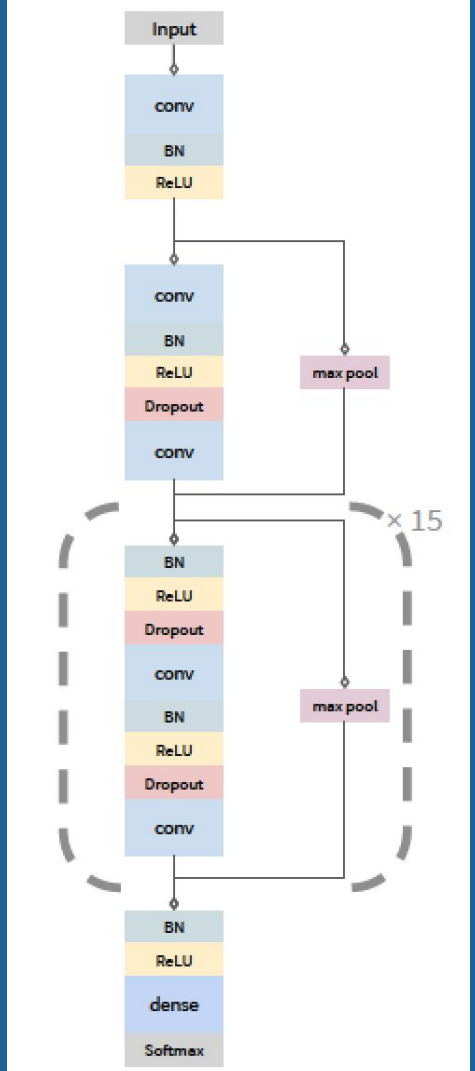
Model



- 34-layer convolutional neural network
- Techniques from automatic speech recognition for processing time-series

- Input: 30-second ECG signal
  - Consecutive one-second samples

- Output: one prediction (class) per second
  - Sequence of 14 output classes:
    - 12 types of arrhythmia, sinus rhythm, or noise



Task: Multi-class, single label classification

Network architecture:

- 33 layers of convolution followed by a fully connected layer and a softmax

- Normalization of input
- Batch normalization before each convolutional layer
- ReLU activation function
- Dropout
- Shortcut connections between layers
- Max pooling
- Categorical cross-entropy loss function

Data

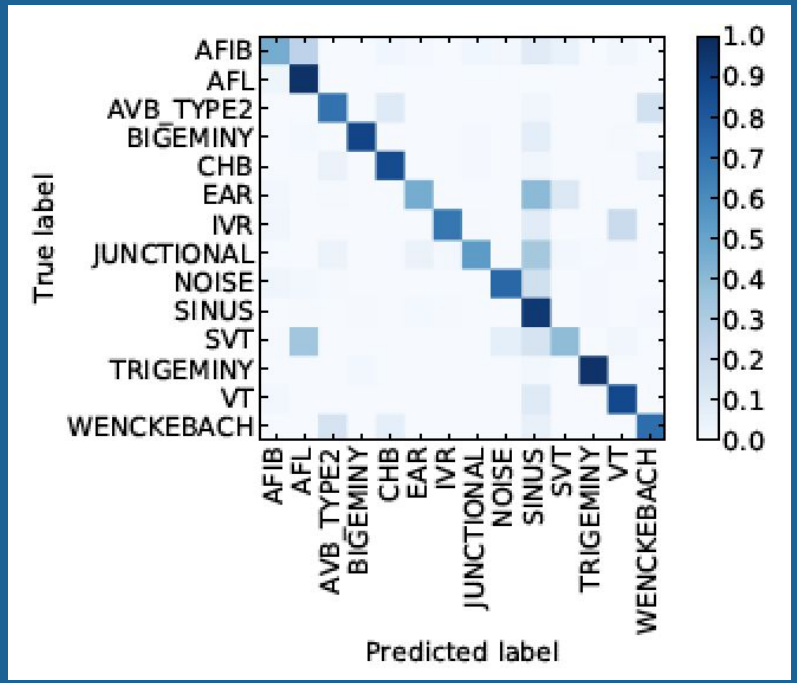
- Training (90%) and Validation (10%)
  - 64,121 ECG records from 29,163 patients
- Testing
  - 336 ECG records from 328 patients
  - Enriched for patients with abnormal rhythms to make class balance more even, and increase likelihood of observing arrhythmia
  - Model predictions were compared to cardiologist performance:
    - 6 individual cardiologists annotated each one-second sample- their performance was averaged
  - Ground truth: a committee of 3 board-certified cardiologists annotated each one-second sample (consensus labeling)

Results (test set): average of cardiologist scores vs. model

- F1 score incorporates sensitivity and PPV
- Sequence level accuracy:
  - Prediction every one second
  - Average overlap between predictions and ground truth
- Set level accuracy:
  - Set of arrhythmias present in each 30-second record
  - Does <u>not</u> penalize for time-misalignment within a 30-second record

	Seq		Set	
	Model	Cardiol.	Model	Cardiol.
Class-level F1 Score				
AFIB	**0.604**	0.515	**0.667**	0.544
AFL	**0.687**	0.635	**0.679**	0.646
AVB_TYPE2	**0.689**	0.535	**0.656**	0.529
BIGEMINY	**0.897**	0.837	**0.870**	0.849
CHB	**0.843**	0.701	**0.852**	0.685
EAR	**0.519**	0.476	**0.571**	0.529
IVR	**0.761**	0.632	**0.774**	0.720
JUNCTIONAL	0.670	**0.684**	**0.783**	0.674
NOISE	**0.823**	0.768	**0.704**	0.689
SINUS	**0.879**	0.847	**0.939**	0.907
SVT	**0.477**	0.449	**0.658**	0.556
TRIGEMINY	**0.908**	0.843	**0.870**	0.816
VT	0.506	**0.566**	0.694	**0.769**
WENCKEBACH	**0.709**	0.593	**0.806**	0.736
Aggregate Results				
Precision (PPV)	**0.800**	0.723	**0.809**	0.763
Recall (Sensitivity)	**0.784**	0.724	**0.827**	0.744
F1	**0.776**	0.719	**0.809**	0.751

Correlations between true labels & model predictions



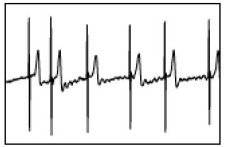
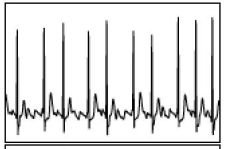
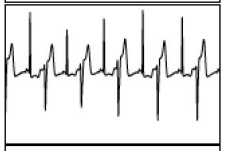
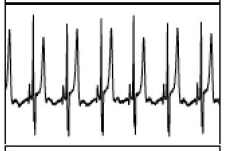
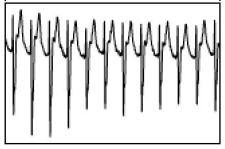
- Model misclassifications indicated in off-diagonal
- Sources of confusion:
  - Similarities in rhythm types
  - Ambiguous location of exact onset/offset of arrhythmia

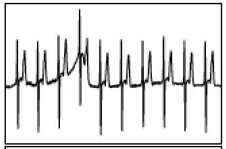
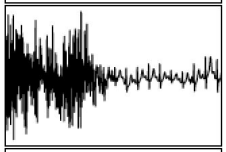
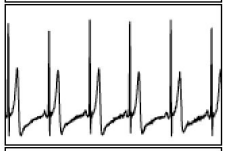
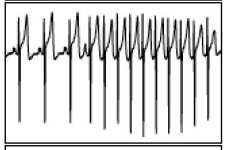
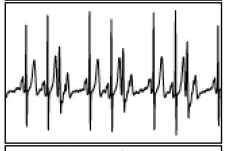
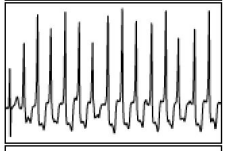
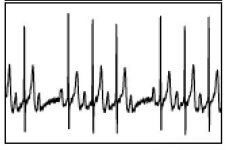
Conclusions

- Model exceeded average cardiologist performance (PPV, sensitivity, F1) in detecting several arrhythmias from single-lead ECG

- Future work:
  - Extensions to other arrhythmias and other heart diseases
  - Data from multiple-lead ECGs

- Clinical implications:
  - Save time for cardiologists, reduce misdiagnoses, implement model where access to cardiologists is limited

Class	Description	Example	Train + Val Patients	Test Patients
AFIB	Atrial Fibrillation		4638	44
AFL	Atrial Flutter		3805	20
AVB_TYPE2	Second degree AV Block Type 2 (Mobitz II)		1905	28
BIGEMINY	Ventricular Bigeminy		2855	22
CHB	Complete Heart Block		843	26
EAR	Ectopic Atrial Rhythm		2623	22
IVR	Idioventricular Rhythm		1962	34

Class	Description	Example	Train + Val Patients	Test Patients
JUNCTIONAL	Junctional Rhythm		2030	36
NOISE	Noise		9940	41
SINUS	Sinus Rhythm		22156	215
SVT	Supraventricular Tachycardia		6301	34
TRIGEMINY	Ventricular Trigeminy		2864	21
VT	Ventricular Tachycardia		4827	17
WENCKEBACH	Wenckebach (Mobitz I)		2051	29

CNNs in Python

MNIST Example

Colab notebook

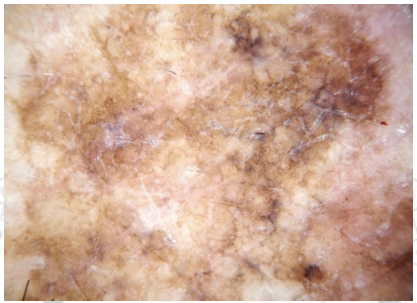
MNIST Example

```
# Define model
model = tf.keras.models.Sequential([
  # Convolutional layer with 32 filters that are 3x3, relu activation function
  tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
  # Pooling layer with 2x2 windows
  tf.keras.layers.MaxPooling2D((2, 2)),

  # Convolutional layer with 64 filters that are 3x3, relu activation function
  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
  # Pooling layer with 2x2 windows
  tf.keras.layers.MaxPooling2D((2, 2)),

  # Convolutional layer with 64 filters that are 3x3, relu activation function
  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),

  # Collapse the 3D tensor
  tf.keras.layers.Flatten(),

  # Fully connected layer with 64 hidden units, relu activation function
  tf.keras.layers.Dense(64, activation='relu'),

  # Softmax output function with 10 classes
  tf.keras.layers.Dense(10, activation='softmax')
])
```

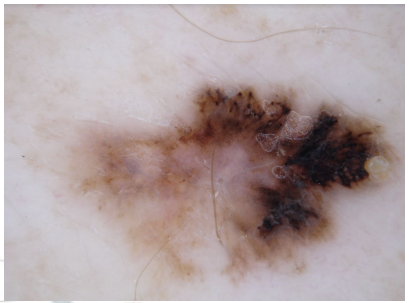
Classifying Skin Lesions

- We'll be using data from the International Skin Imaging Collaboration: Melanoma Project
- We'll be classifying images as malignant or benign
- The overarching goal of the ISIC Melanoma Project is to support efforts to reduce melanoma-related deaths and unnecessary biopsies by improving the accuracy and efficiency of melanoma early detection

Malignant



Malignant



Benign

Benign

Classifying Skin Lesions

- This archive contains 23k images of classified skin lesions. It contains both malignant and benign examples
  - We'll be using a fraction of this
- Each example contains the image of the lesion, meta data regarding the lesion (including classification and segmentation) and meta data regarding the patient
- The data can be viewed in this link (in the gallery section)
- It can be downloaded through the site or by using this repository

Classification Skin Lesions

- The subsample of the data is available in a Google Drive folder
- You can access it with this code and notebook
- You can also download the images to your machine if you would like
  - There are zip files available on canvas
- We'll start with creating a simple CNN

```python
# Define model
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Flatten(),

  tf.keras.layers.Dense(512, activation='relu'),

  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

```python
# Define model
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Flatten(),

  tf.keras.layers.Dense(512, activation='relu'),

  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Convolution and pooling layers - notice how the output size decreases with each layer. Remember what applying a filter, padding, and/or strides does to an input.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

```python
# Define model
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Flatten(),

  tf.keras.layers.Dense(512, activation='relu'),

  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

We finish the network with 1 hidden dense layer and 1 output layer. Note that most of the parameters in the model come from the hidden dense layer and not the convolution or pooling layers.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

```python
# Define model
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D((2, 2)),

  tf.keras.layers.Flatten(),

  tf.keras.layers.Dense(512, activation='relu'),

  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

Total # of parameters that need to be learned by the network

```
from keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
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')
```

We first scale the data to get values between 0 and 1.

Then we transform the images to be 150x150 pixels in size (this is arbitrary), declare a batch size of 20 (this is also arbitrary), and declare the class mode (i.e. the type of classification we want to do)

```
for data_batch, labels_batch in train_generator:
    print('data batch shape:', data_batch.shape)
    print('labels batch shape:', labels_batch.shape)
    break
```

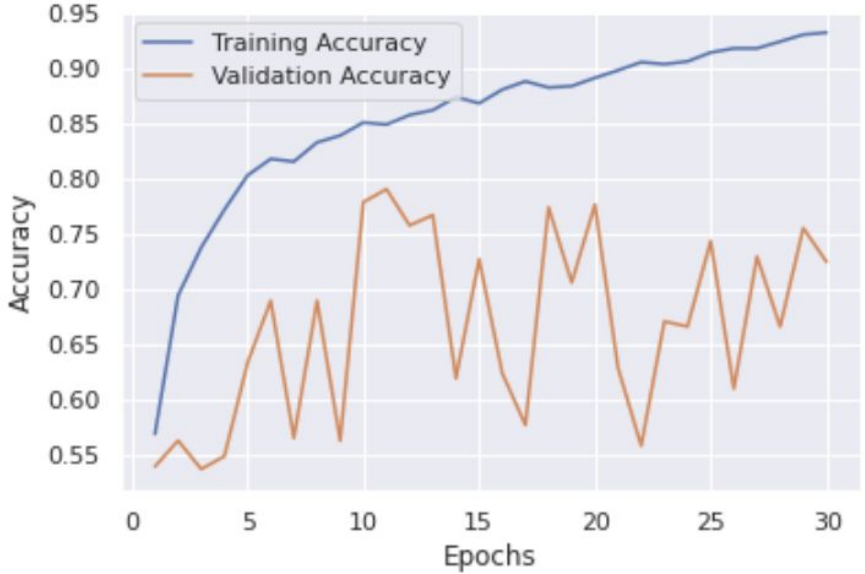
```
data batch shape: (20, 150, 150, 3)
labels batch shape: (20,)
```

We can see the shape of the images: they are in batches of 20, with each image being represented by 3, 150x150 tensors: one for R, one for G and one for B color “channels”.

```
history = model.fit(
      train_generator,
      steps_per_epoch = 81, # ceil(1609/20)
      epochs = 30,
      validation_data = validation_generator,
      validation_steps = 22) # ceil(426/20)
```

The number of training examples divided by the batch size, i.e. how many batches we need to go through until the model sees all training data. For both the training and validation sets.


Loss
Epochs
Training Loss
Validation Loss
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
5
10
15
20
25
30


Accuracy
Epochs
Training Accuracy
Validation Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
5
10
15
20
25
30

Data Augmentation

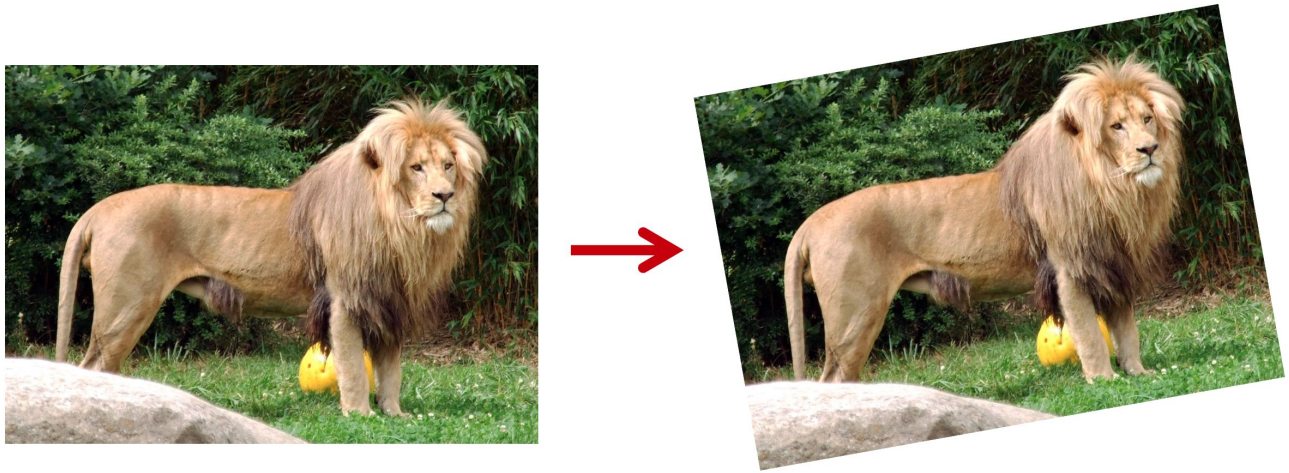
- As we have seen, overfitting is caused by having too few training examples to learn from
- Data augmentation generates more training data from existing training examples by **augmenting** the samples via a number of random transformations
- These transformations should yield believable images

Data Augmentation

- Types of augmentation:
  - Rotation
  - Horizontal/vertical flip
  - Random crops/scales
    - Zoom
    - Width or height shifts
  - Shearing
  - Brightness, contrast, saturation
  - Lens distortions

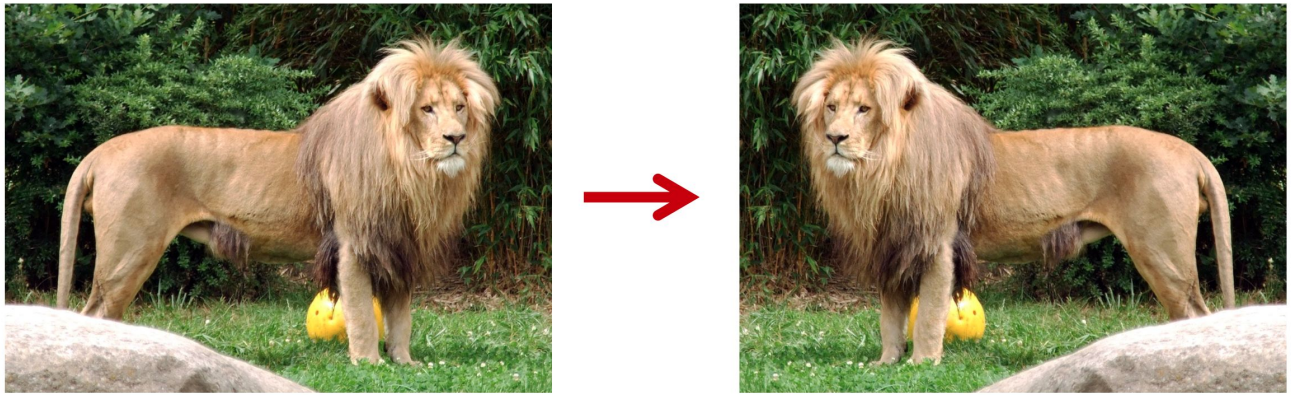
Types of data augmentation

1. Rotations



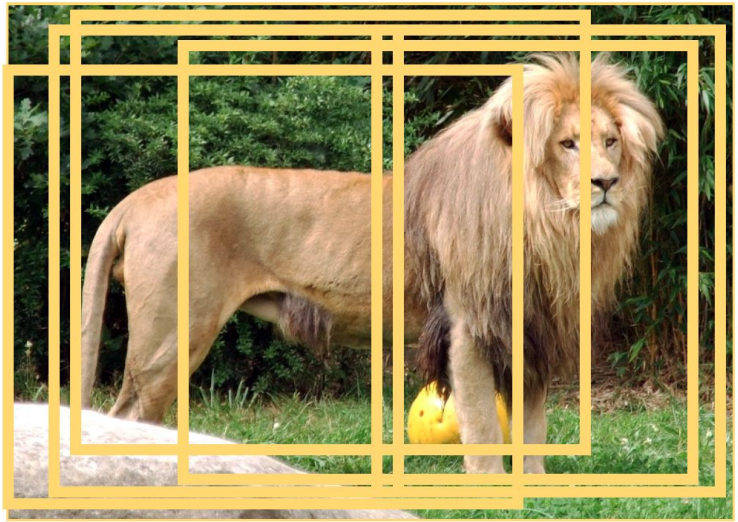
Types of data augmentation

2. Horizontal/Vertical Flips



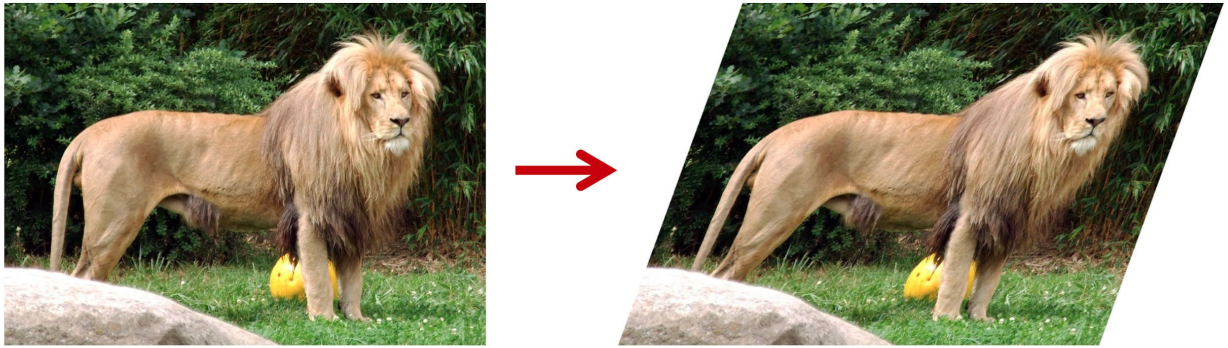
Types of data augmentation

3. Random crops/scales



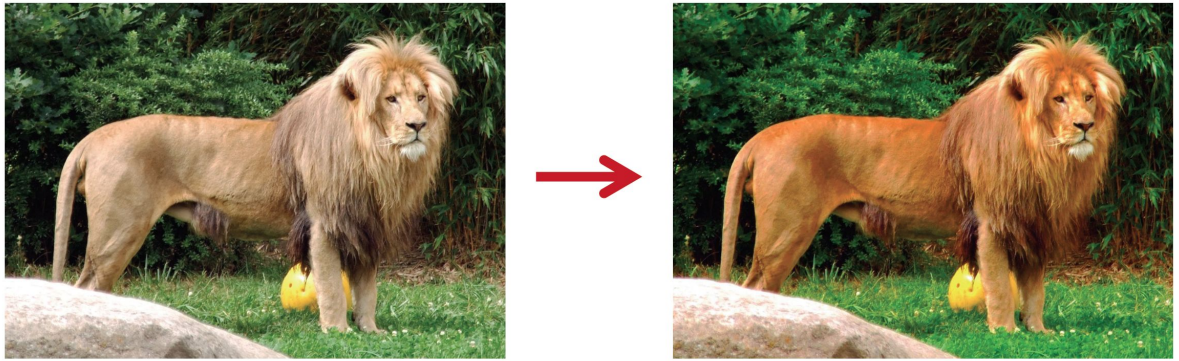
Types of data augmentation

4. Shearing



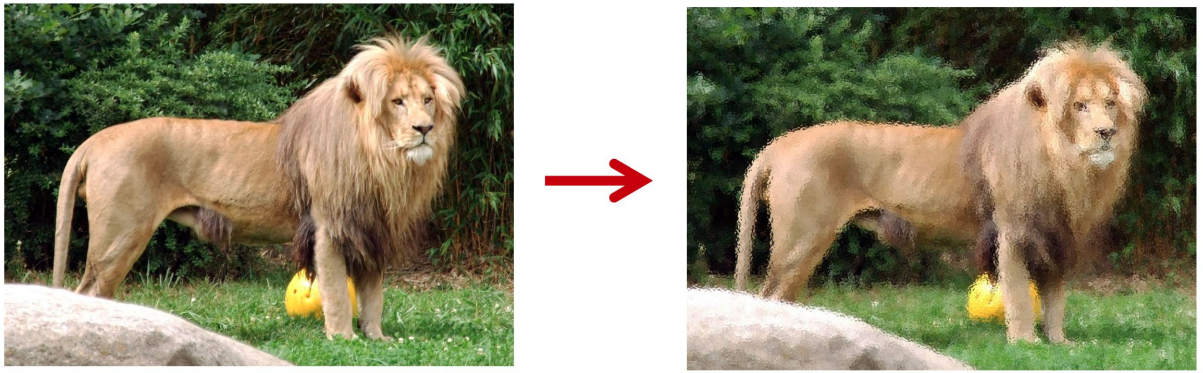
Types of data augmentation

5. Brightness, contrast, saturation



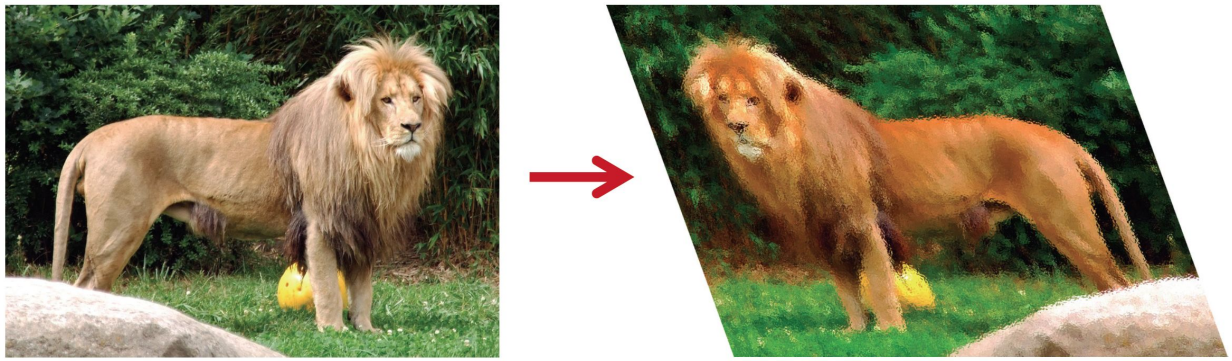
Types of data augmentation

6. Lens distortions



Types of data augmentation

7. Combinations of the above



Data Augmentation

- If you train a network using data augmentation, it will never see the same input twice, but the inputs will still be heavily correlated
  - You're remixing known information, not producing new information
- May not completely escape overfitting due to this correlation
- Adding dropout can also help

Data Augmentation in Keras

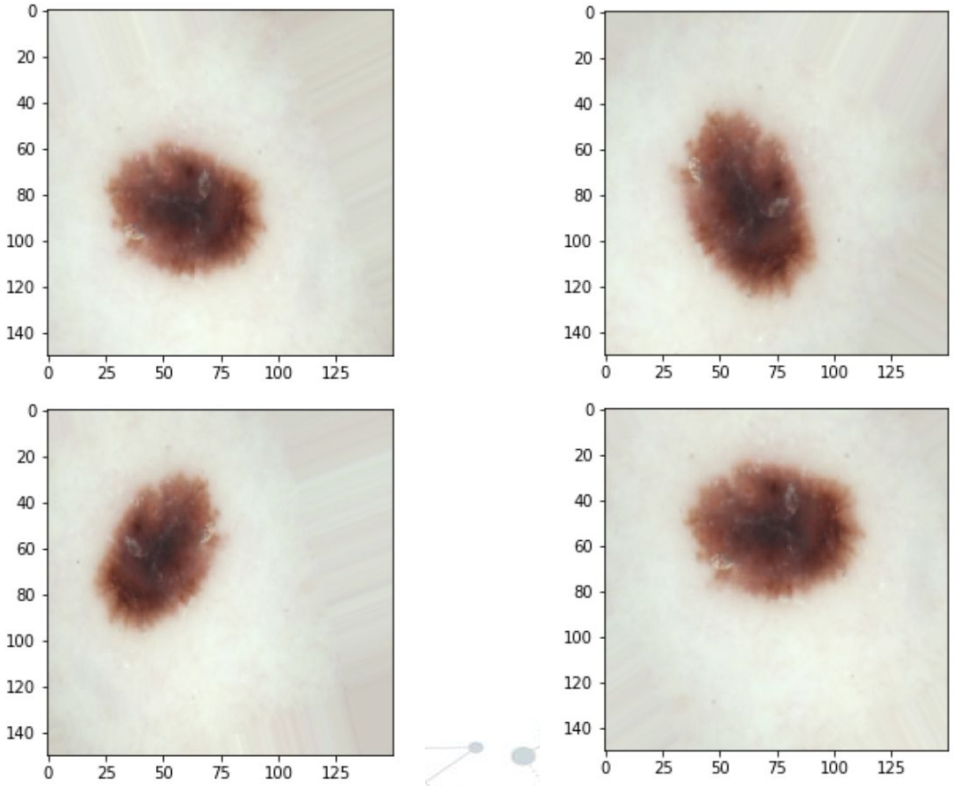
```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
      rotation_range = 40,
      width_shift_range = 0.2,
      height_shift_range = 0.2,
      shear_range = 0.2,
      zoom_range = 0.2,
      horizontal_flip = True,
      fill_mode = 'nearest')
```

You can create your own data generator with any specifications you'd like. The values chosen here are arbitrary.

You can check out the Keras documentation to see all of the available options and values each type of augmentation type can take.

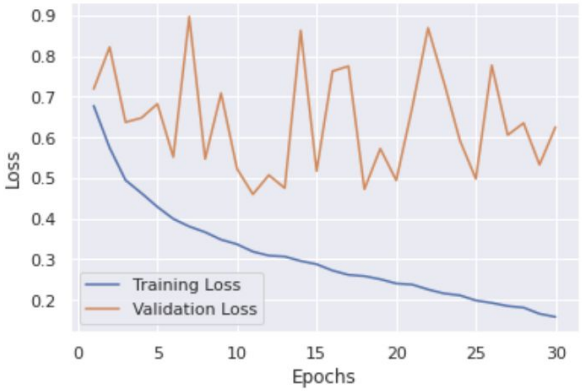
Note that only your training data should be augmented - not the test or validation sets. The point of augmentation is to "increase" your training set size.

Data Augmentation in Keras



Data Augmentation in Keras

Without augmentation



With augmentation