EECE-4710 "IoT and TinyML"

Preventing Overfitting

Cris Ababei

MARQUETTE UNIVERSITY

BE THE DIFFERENCE.

1

Preventing Overfitting

+Data

+Data

+Data

+Data

But what to do if we do not have more data?

- Data Augmentation (artificial)
- Transfer Learning
- Early Stopping
- Dropout Regularization

2

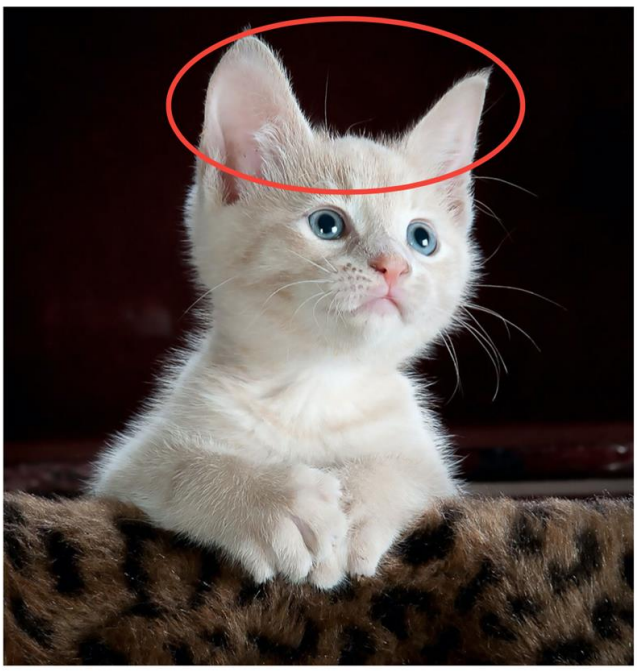
Preventing Overfitting

More Data, Data Augmentation (artificial)

Overfitting generally occurs when there are a small number of training examples. Data augmentation takes the approach of generating additional training data from your existing examples by augmenting them using random transformations that yield believable-looking images. This helps expose the model to more aspects of the data and generalize better.

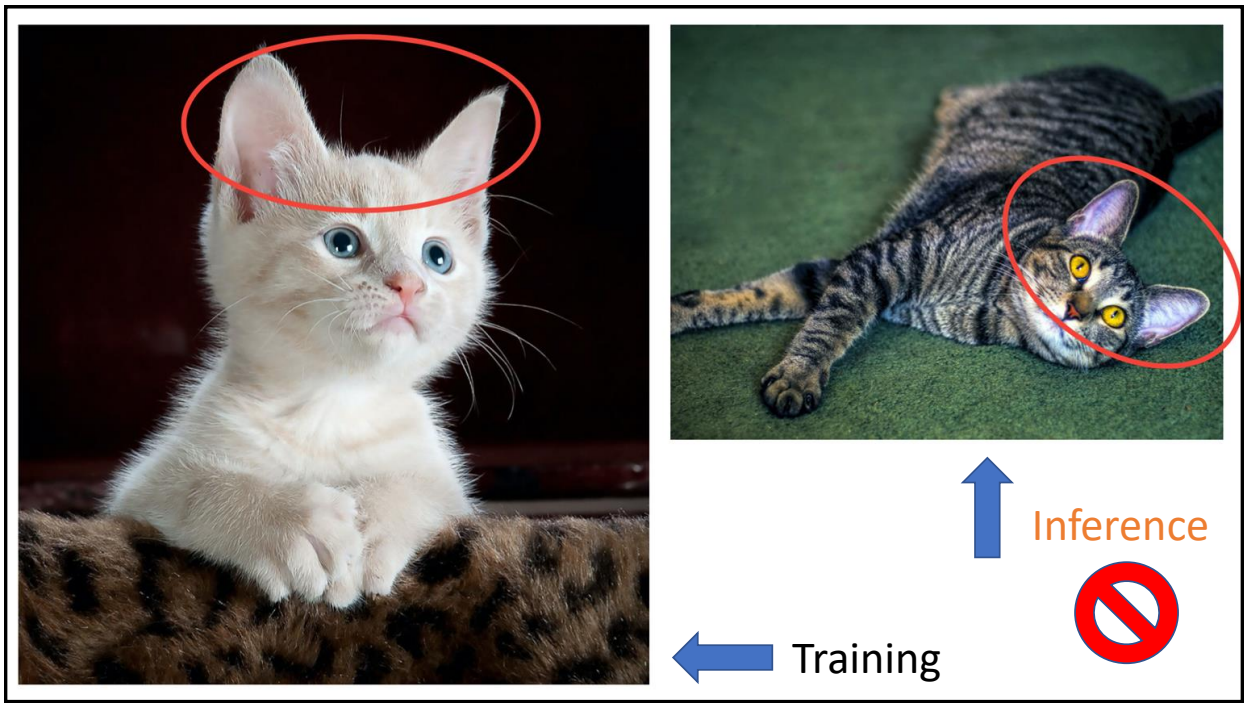
3

3

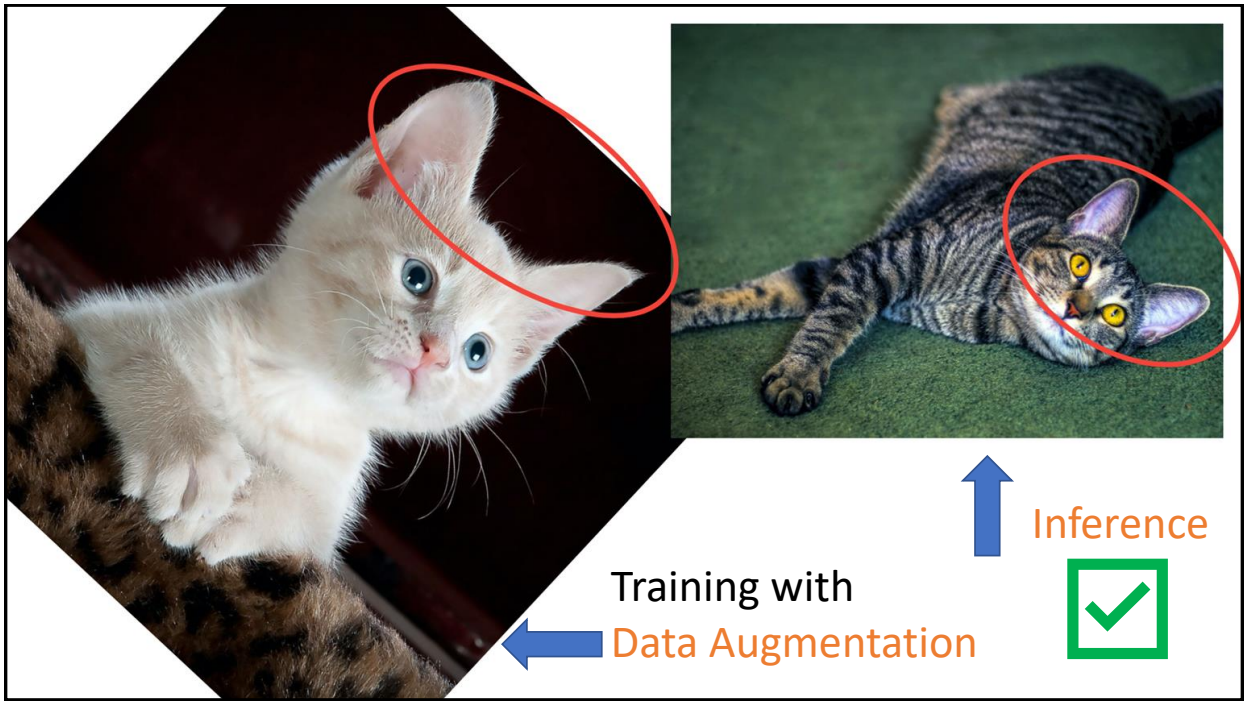


Training

4


Inference
Training

5


Inference
Training with
Data Augmentation

6

Using Keras Preprocessing Layers

```
data_augmentation = tf.keras.Sequential([
  layers.RandomFlip("horizontal_and_vertical"),
  layers.RandomRotation(0.2),
])
```

```
plt.figure(figsize=(10, 10))
for i in range(9):
  augmented_image = data_augmentation(image)
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(augmented_image[0])
  plt.axis("off")
```

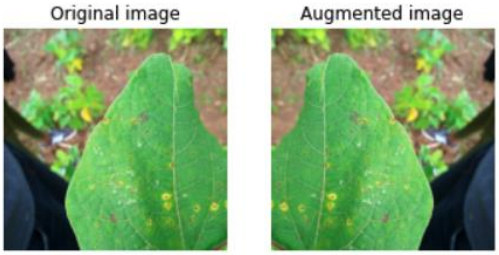
A variety of preprocessing layers you can use for data augmentation including:

- tf.keras.layers.RandomContrast
- tf.keras.layers.RandomCrop
- tf.keras.layers.RandomZoom
- ...

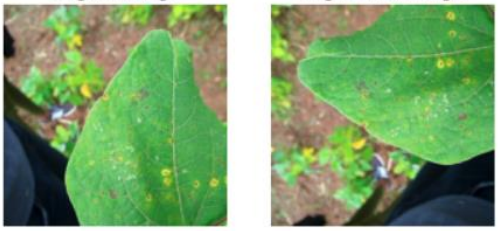
7

Using tf.image

```
flipped = tf.image.flip_left_right(image)
visualize(image, flipped)
```



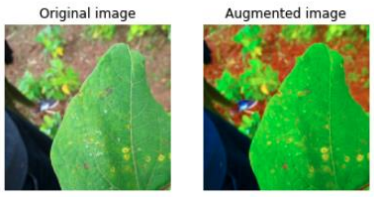

```
rotated = tf.image.rot90(image)
visualize(image, rotated)
```



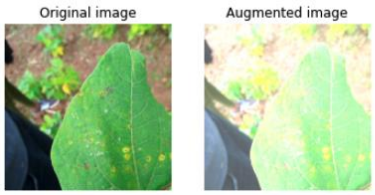
8

Using tf.image

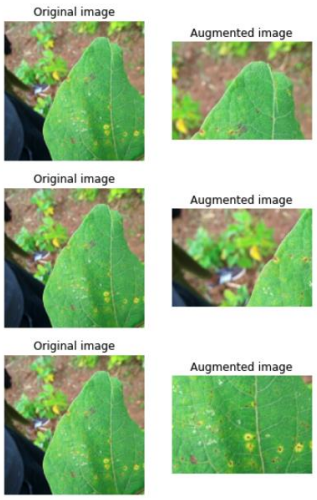
```
saturated = tf.image.adjust_saturation(image, 3)
visualize(image, saturated)
```




```
bright = tf.image.adjust_brightness(image, 0.4)
visualize(image, bright)
```




```
for i in range(3):
  seed = (i, 0)  # tuple of size (2,)
  stateless_random_crop = tf.image.stateless_random_crop(
      image, size=[210, 300, 3], seed=seed)
  visualize(image, stateless_random_crop)
```



9

Data Augmentation

Code Time!

IESTI01_data_augmentation.ipynb

10

10

Preventing Overfitting

Early Stopping & Dropout Regularization

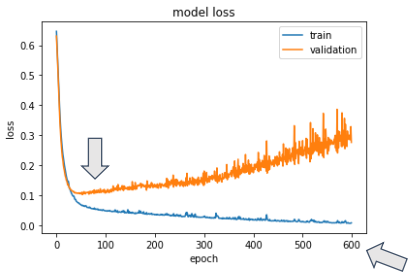
11

Early Stopping

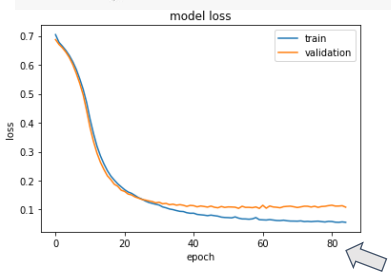
```
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss',
                           mode='min',
                           verbose=1,
                           patience=25)
```

```
history = model.fit(X_train,
                    y_train,
                    epochs=600,
                    validation_data=(X_test, y_test),
                    verbose=1
                    )
```



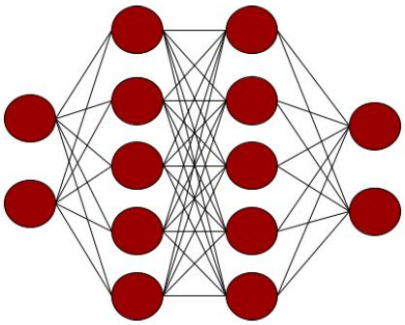

```
history = model.fit(x=X_train,
                    y=y_train,
                    epochs=600,
                    validation_data=(X_test, y_test),
                    verbose=1,
                    callbacks=[early_stop]
                    )
```



12

Dropout Regularization

```
model = tf.keras.models.Sequential([
                    tf.keras.layers.Flatten(input_shape=(28,28)),
                    tf.keras.layers.Dense(256, activation=tf.nn.relu),
                    tf.keras.layers.Dense(128, activation=tf.nn.relu),
                    tf.keras.layers.Dense(64, activation=tf.nn.relu),
                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```



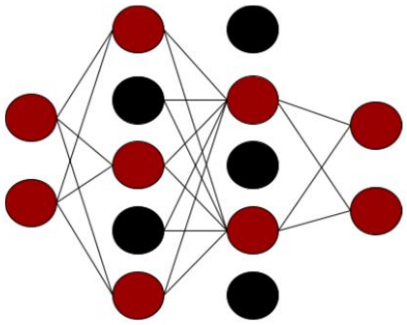
Fashion MNIST Dataset

- 20 Epochs
- 94.0% Accuracy on Train Data
- 88.5% Accuracy on Validation Data

13

Dropout Regularization

```
model = tf.keras.models.Sequential([
                    tf.keras.layers.Flatten(input_shape=(28,28)),
                    tf.keras.layers.Dense(256, activation=tf.nn.relu),
                    tf.keras.layers.Dropout(0.2),
                    tf.keras.layers.Dense(128, activation=tf.nn.relu),
                    tf.keras.layers.Dropout(0.2),
                    tf.keras.layers.Dense(64, activation=tf.nn.relu),
                    tf.keras.layers.Dropout(0.2),
                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```



Fashion MNIST Dataset

- 20 Epochs
- 89.5% Accuracy on Train Data
- 88.3% Accuracy on Validation Data

Removing a random number of neurons and connections (in this example, 20%), reduces the chances of the neurons becoming overspecialized and the model will generalize better, reducing the overfit.

14

Breast Cancer BINARY Classification
Wisconsin Diagnostic Breast Cancer (WDBC)
Code Time!

Breast_Cancer_Classification.ipynb

15

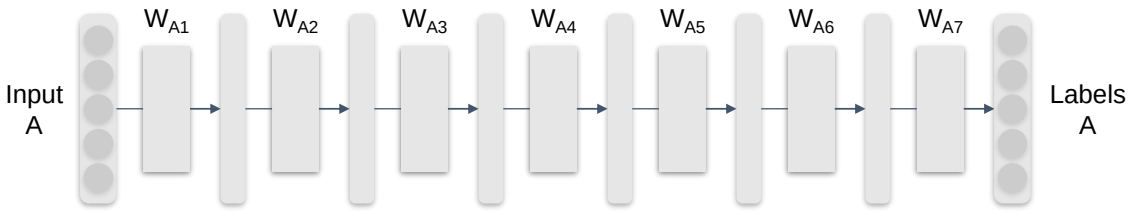
15

Preventing Overfitting
Transfer Learning

16

16

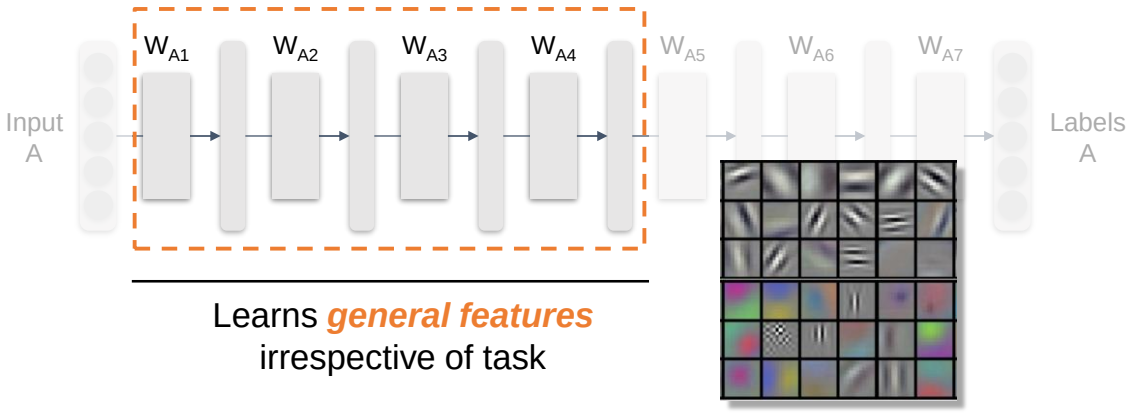
End Result of Training



Result of training is to learn the weights of neural network model.

17

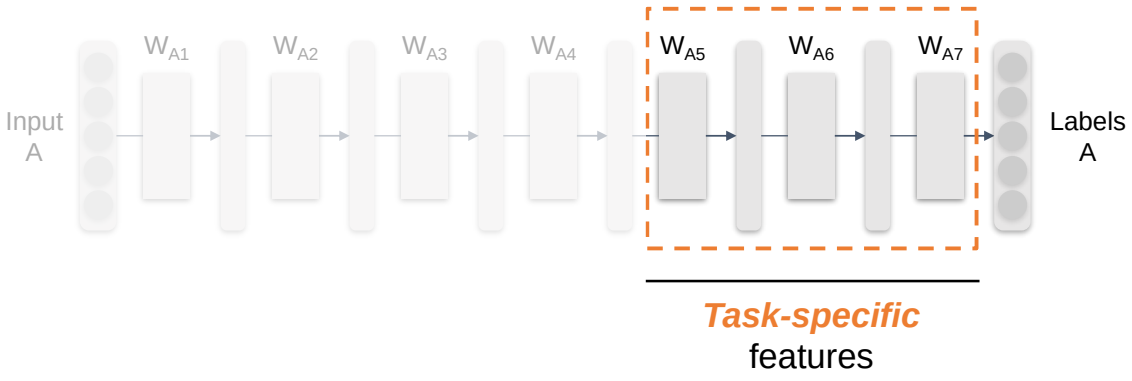
End Result of Training



Learns ***general features***
irrespective of task

18

End Result of Training

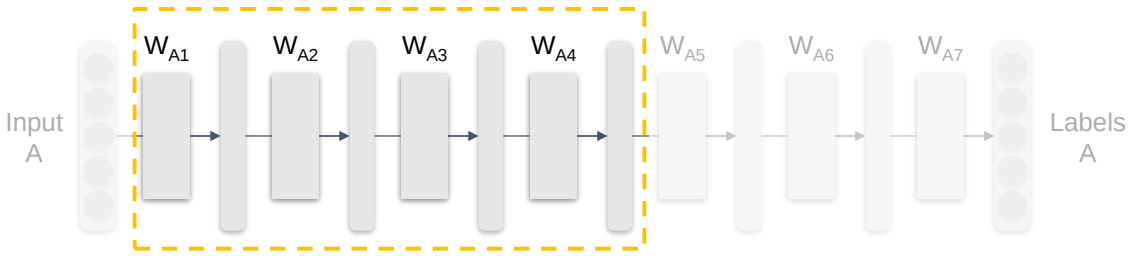


Task-specific
features

19

Transfer Learning

Reuse (freeze general feature extraction)

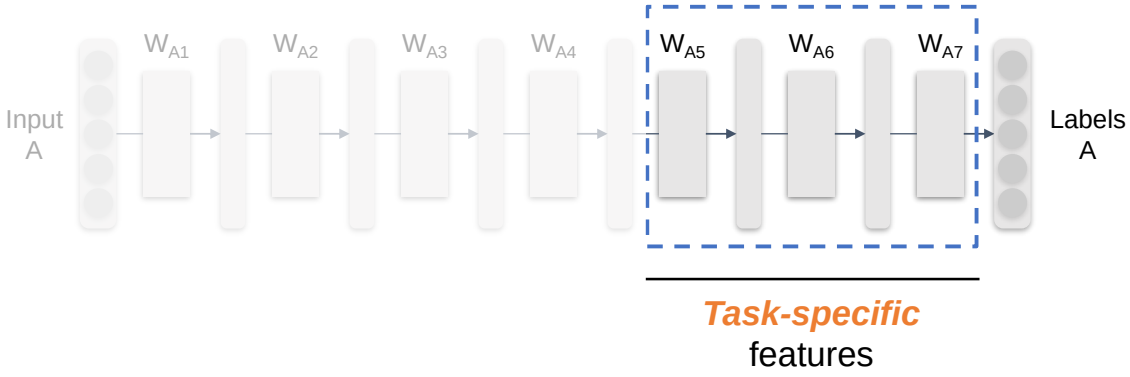


Learns ***general features***
irrespective of task

20

Transfer Learning

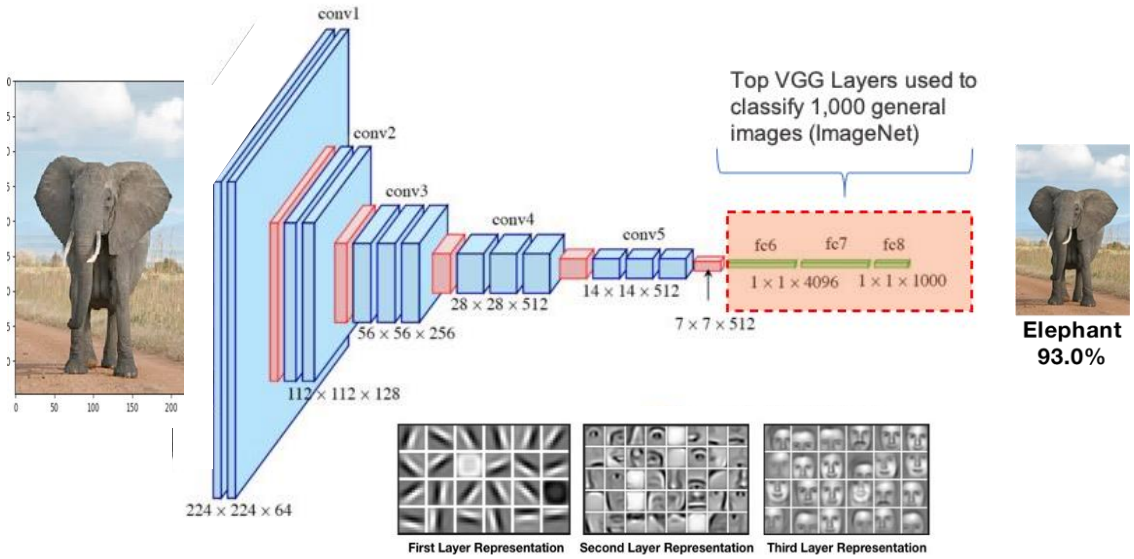
Train **only** last few layers



Task-specific features

21

Example: VGG-16 Convolutional Neural Network Model



22

VGG-16

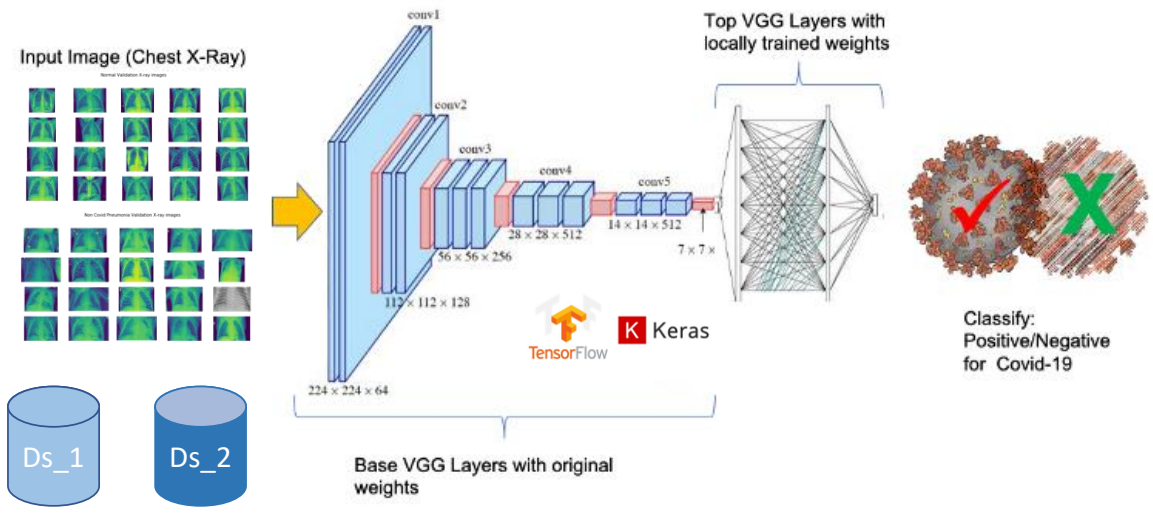
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual computer vision competition. Each year, teams compete on two tasks:
  - The first is to detect objects within an image coming from 200 classes, which is called **object localization**.
  - The second is to classify images, each labeled with one of 1000 categories, which is **called image classification**.
- VGG-16 was proposed by Karen Simonyan and Andrew Zisserman of the Visual Geometry Group Lab of Oxford University in 2014 in the paper "VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION".
  - This model won 1st and 2nd place in the above categories in the 2014 ILSVRC challenge.

https://www.mygreatlearning.com/blog/introduction-to-vgg16/
https://www.geeksforgeeks.org/vgg-16-cnn-model/

23

23

Training the model (Transfer Learning)



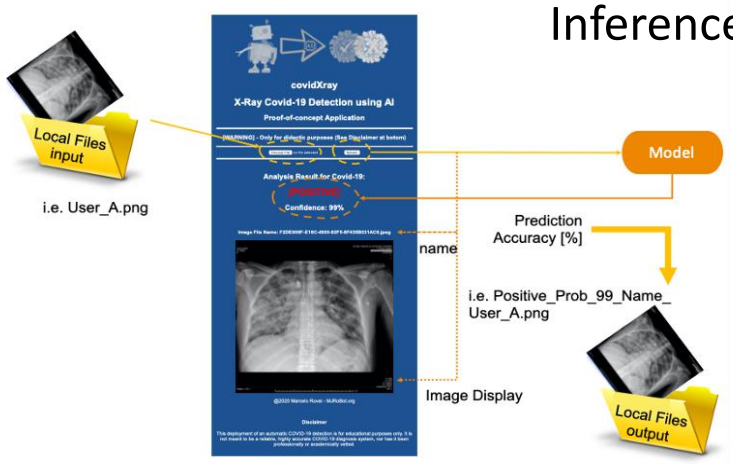
24

covidXray

Detecting Covid-19 in Chest X-Ray images

https://github.com/Mjrovai/covid19Xray

Inference



25

Credits

- A previous edition of this course was developed in collaboration with Dr. Susan C. Schneider of Marquette University.
- We are very grateful and thank all the following professors, researchers, and practitioners for jump-starting courses on TinyML and for sharing their teaching materials:

- Prof. Marcelo Rovai - TinyML - Machine Learning for Embedding Devices, UNIFEI
  - https://github.com/Mjrovai/UNIFEI-IESTI01-TinyML-2022.1
- Prof. Vijay Janapa Reddi - CS249r: Tiny Machine Learning, Applied Machine Learning on Embedded IoT Devices, Harvard
  - https://sites.google.com/g.harvard.edu/tinyml/home
- Prof. Rahul Mangharam – ESE3600: Tiny Machine Learning, Univ. of Pennsylvania
  - https://tinyml.seas.upenn.edu/#
- Prof. Brian Plancher - Harvard CS249r: Tiny Machine Learning (TinyML), Barnard College, Columbia University
  - https://a2r-lab.org/courses/cs249r_tinyml/

26

26

References

- Additional references from where information and other teaching materials were gathered include:

- Applications & Deploy textbook: “TinyML” by Pete Warden, Daniel Situnayake
  - https://www.oreilly.com/library/view/tinyml/9781492052036/
- Deploy textbook “TinyML Cookbook” by Gian Marco Iodice
  - https://github.com/PacktPublishing/TinyML-Cookbook
- Jason Brownlee
  - https://machinelearningmastery.com/
- TinyMLedu
  - https://tinyml.seas.harvard.edu/
- Professional Certificate in Tiny Machine Learning (TinyML) – edX/Harvard
  - https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning
- Introduction to Embedded Machine Learning - Coursera/Edge Impulse
  - https://www.coursera.org/learn/introduction-to-embedded-machine-learning
- Computer Vision with Embedded Machine Learning - Coursera/Edge Impulse
  - https://www.coursera.org/learn/computer-vision-with-embedded-machine-learning