ML Project
Face Mask Detection

Group 3

Team Member: Huishan Zhang, Linjing Li, Xi Fei, Zhujun Tian

AGENDA

Part 1 Project Introduction

Part 2 Dataset Characteristics

Part 3 Convolutional Neural Network

Part 4 Data Processing and Training

Part 5 Testing and Evaluation

Part 6 Conclusion

Part 1 Project Introduction

Project introduction

Project introduction

- Face Mask-wearing is an efficient way to help control Covid-19 outbreak
- As Face Mask is became essential for everyone while roaming outside, identifying mask wear help monitor public behavior and contribute towards constraining the COVID-19 pandemic.

Part 2 Dataset Characteristics

Dataset Characteristics

Data Explorer

2.58 GB

- Medical mask
  - Medical mask
    - Medical Mask
      - annotations
      - images
      - meta.json
  - submission.csv
  - train.csv

annotations (4326 files)

1801.jpg.json 208 B	1802.jpg.json 361 B	1803.jpg.json 360 B	1804.jpg.json 362 B
1805.jpg.json 502 B	1806.jpg.json 510 B	1807.jpg.json 358 B	1808.jpg.json 358 B

images (6024 files)

train.csv (571.71 KB)

Detail Compact Column 6 of 6 columns

name	x1	x2	y1	y2	classname
2756.png	69	126	294	392	face_with_mask

Dataset Characteristics

- **train.csv:** take the first record from meta_data(train.csv); then extract all the records with the same name (‘2756.png’) from meta_data(train.csv)
- **images file:** plot the images with the same name (‘2756.png’)
- **annotations file:** detail information regrading FimeName ‘2756.png’

train.csv

```
meta_name=meta_data["name"][0]
meta_name
```

```
'2756.png'
```

```
meta_data[meta_data["name"]==meta_name]
```

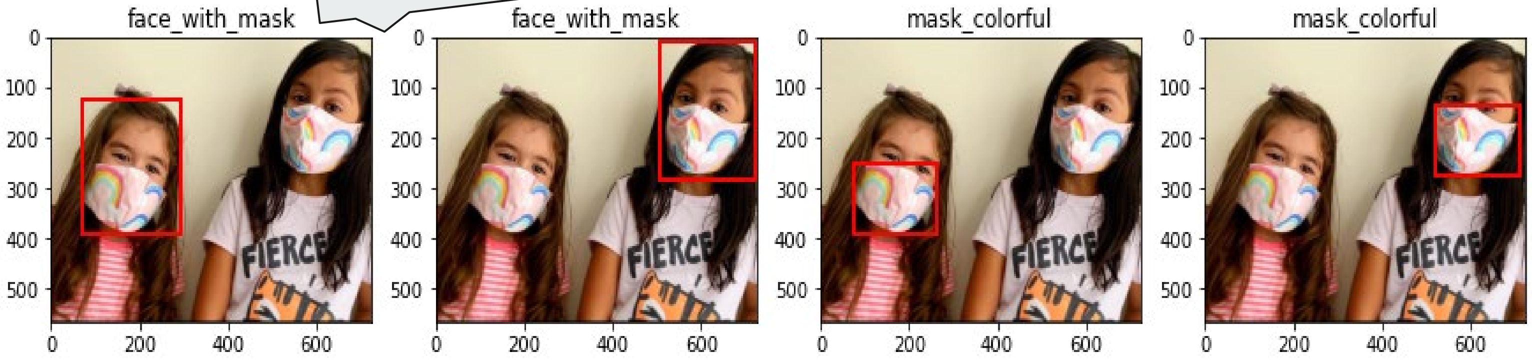
	name	x1	x2	y1	y2	classname
0	2756.png	69	126	294	392	face_with_mask
1	2756.png	505	10	723	283	face_with_mask
2	2756.png	75	252	264	390	mask_colorful
3	2756.png	521	136	711	277	mask_colorful

annotations

```
jsonfiles= []
for i in os.listdir(directory):
    jsonfiles.append(getJSON(os.path.join(directory,i)))
jsonfiles[878]
```


```
{'FileName': '2756.png',
 'NumOfAnno': 4,
 'Annotations': [{'isProtected': False,
   'ID': 598039385457921920,
   'BoundingBox': [69, 126, 294, 392],
   'classname': 'face_with_mask',
   'Confidence': 1,
   'Attributes': {}},
  {'isProtected': False,
   'ID': 702410169516003712,
   'BoundingBox': [505, 10, 723, 283],
   'classname': 'face_with_mask',
   'Confidence': 1,
   'Attributes': {}},
  {'isProtected': False,
   'ID': 152706552218535680,
   'BoundingBox': [75, 252, 264, 390],
   'classname': 'mask_colorful',
   'Confidence': 1,
   'Attributes': {}},
  {'isProtected': False,
   'ID': 3262639018530855,
   'BoundingBox': [521, 136, 711, 277],
   'classname': 'mask_colorful',
   'Confidence': 1,
   'Attributes': {}}]}
```


images



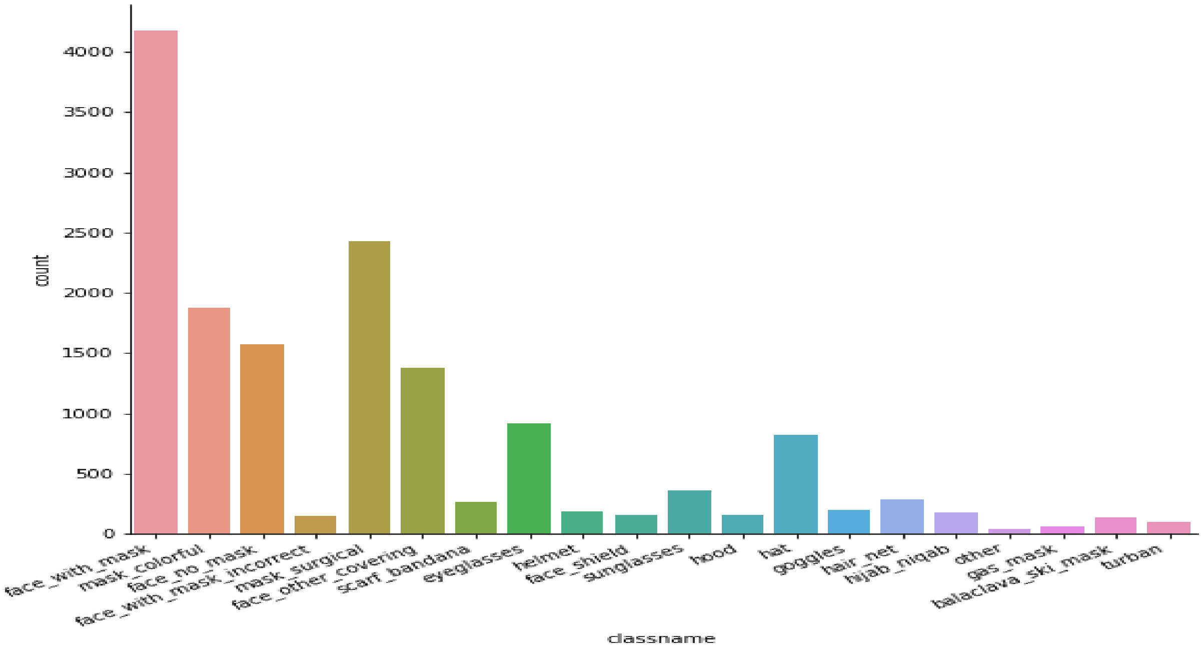
Dataset Characteristics - train.csv(meta_data)

```
meta_data.head()
```

	name	x1	x2	y1	y2	classname
0	2756.png	69	126	294	392	face_with_mask
1	2756.png	505	10	723	283	face_with_mask
2	2756.png	75	252	264	390	mask_colorful
3	2756.png	521	136	711	277	mask_colorful
4	6098.jpg	360	85	728	653	face_no_mask

```
print(len(meta_data))
```

```
15412
```




```
## Counts of face_with_mask and face_no_mask
face_with_mask=train_data[train_data["classname"]=="face_with_mask"]
face_no_mask=train_data[train_data["classname"]=="face_no_mask"]
print("count of face with mask: "+str(len(face_with_mask))+"\ncount of face no mask: "+str(len(face_no_mask)))
```

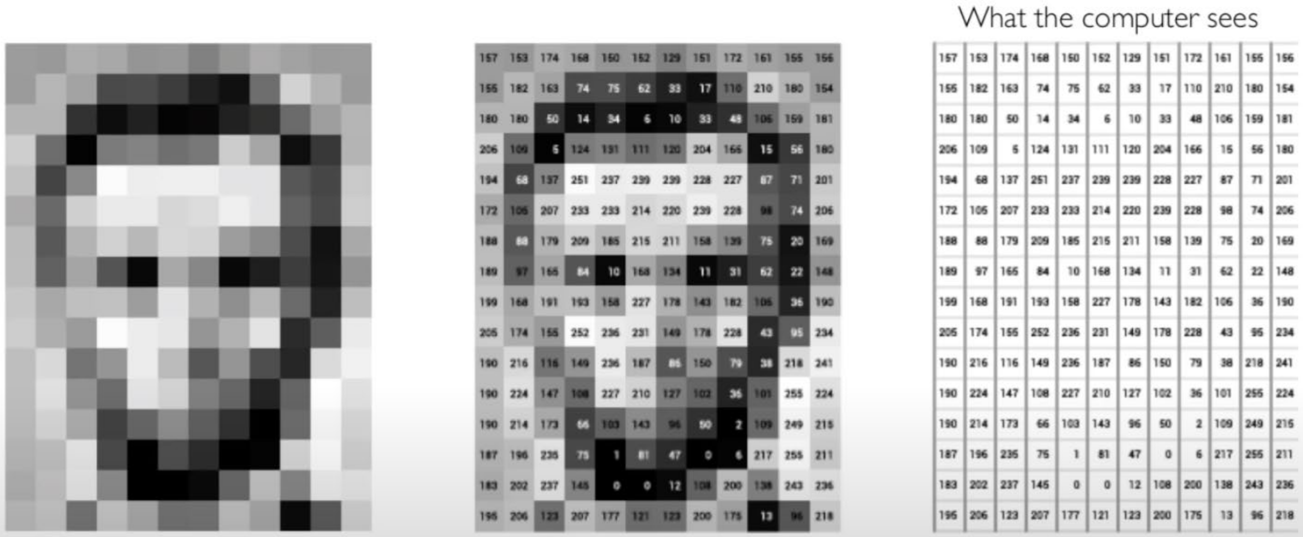
```
count of face with mask: 4180
count of face no mask: 1569
```

Part 3
Convolutional Neural Network

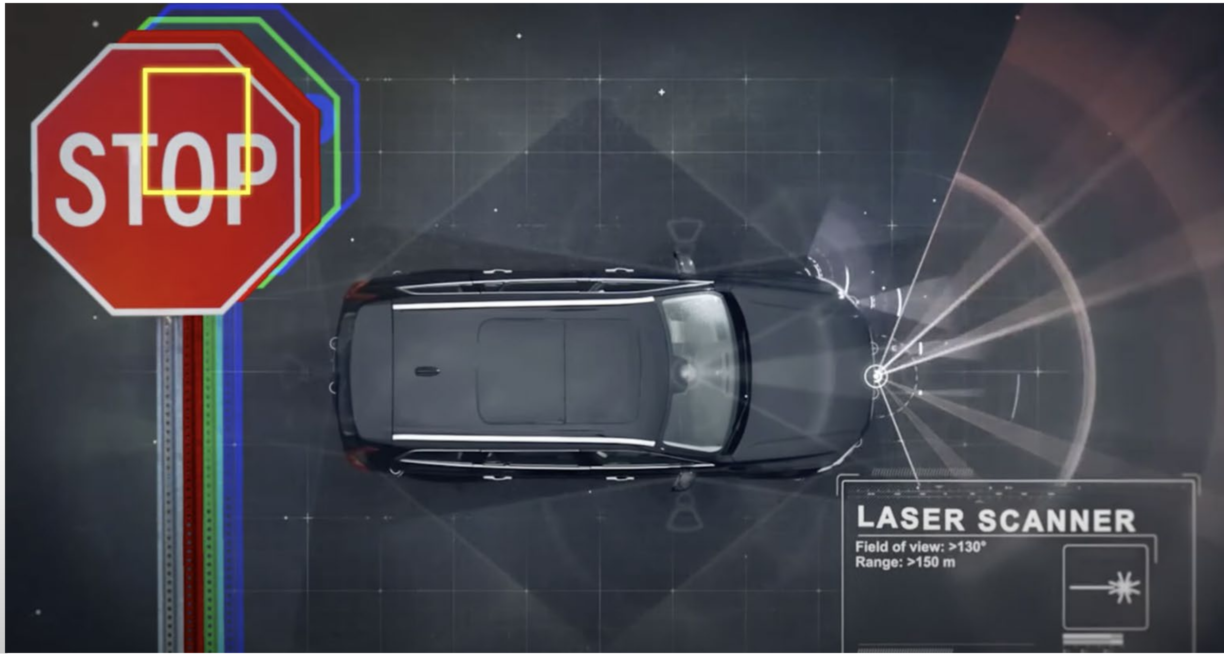
Convolutional Neural Network(CNN)

Technology that teaches computer to 'see'

Application: Image Recognition, Autonomous Driving, Teach Computer to Play Video Games/ Cooking...

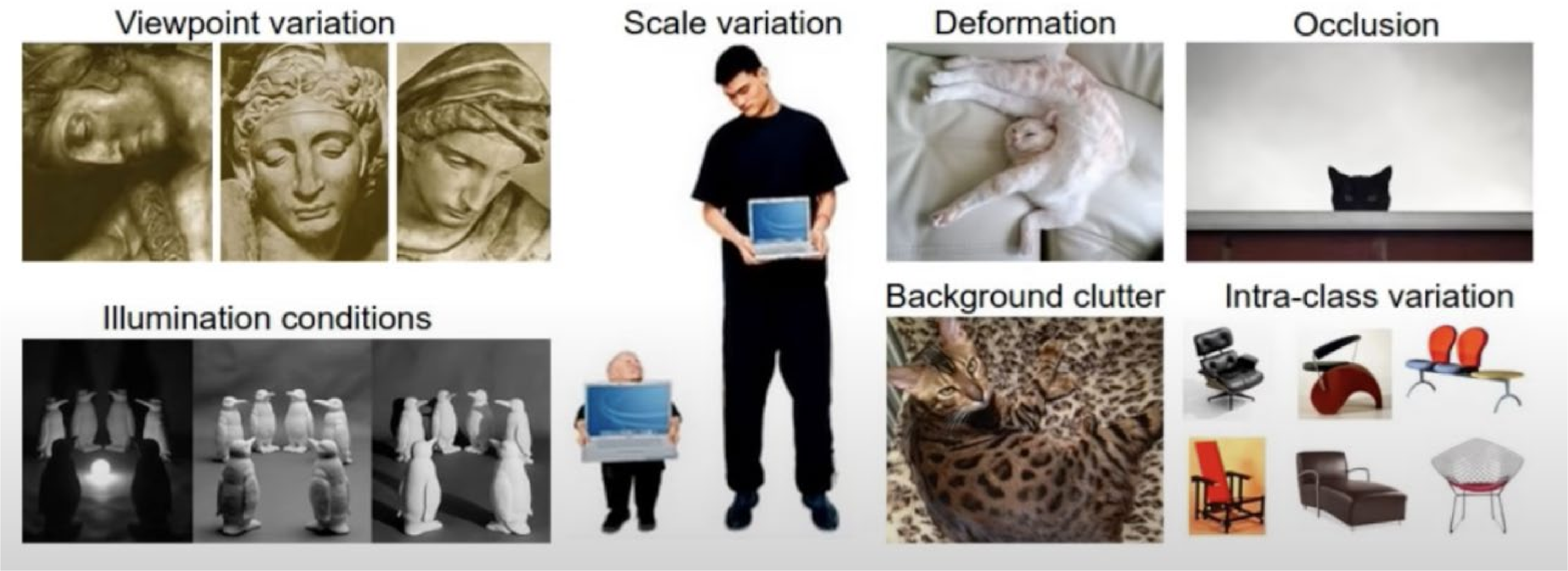


An image is just a matrix of numbers [0,255]!
i.e., 1080x1080x3 for an RGB image



Convolutional Neural Network(CNN)

What make the recognition so hard for the computer?



Solution?

Feature Detector/Filter/Kernel

CNN Key Concepts and Architecture -- Filtering

Filters are common concept in the image processing field. They are usually number matrix that can detect horizontal, vertical, edges… of images. But in CNNs, filters are not defined. The value of each filter is learned during the training process.

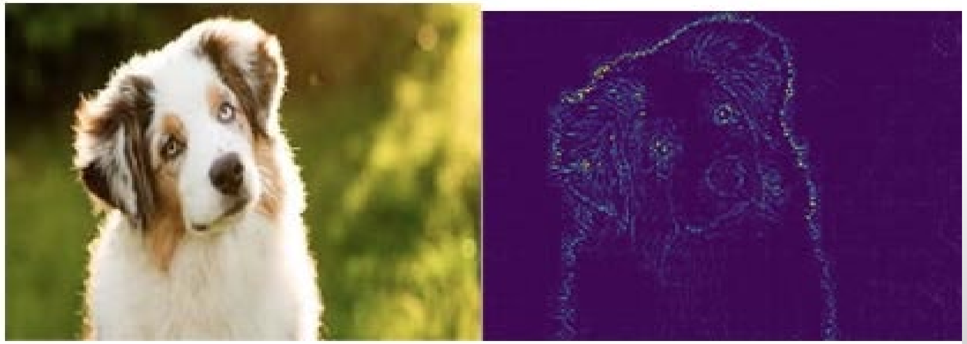
By being able to learn the values of different filters, CNNs can find more meaning from images that humans and human designed filters might not be able to find.

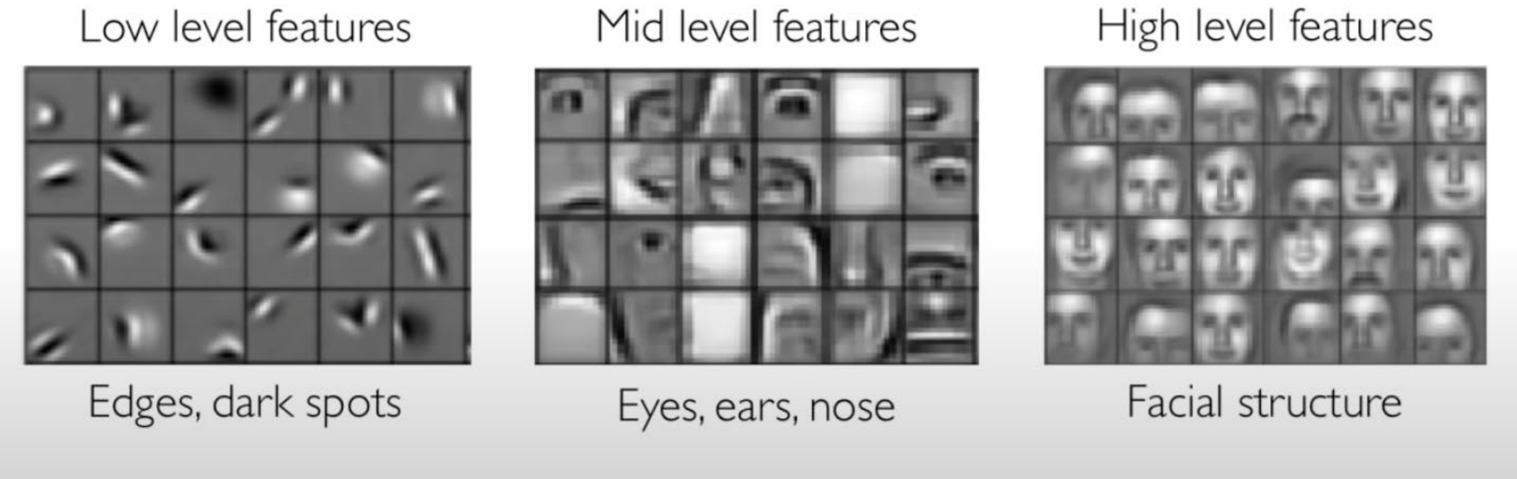
-1	-2	-1
0	0	0
1	2	1

Horizontal

-1	0	1
-2	0	2
-1	0	1

Vertical

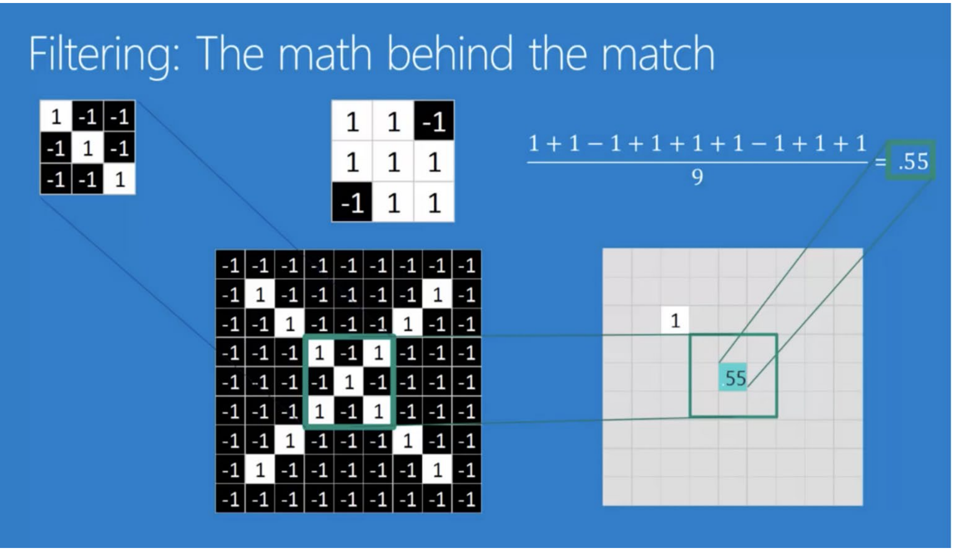


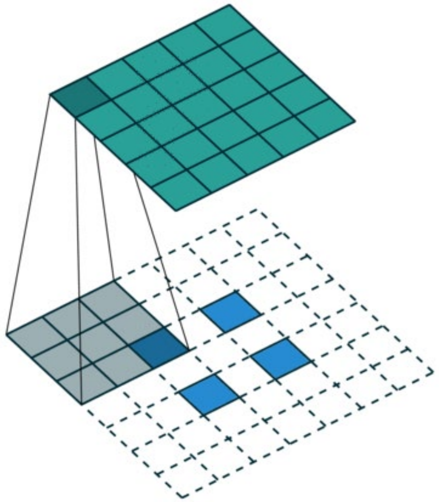


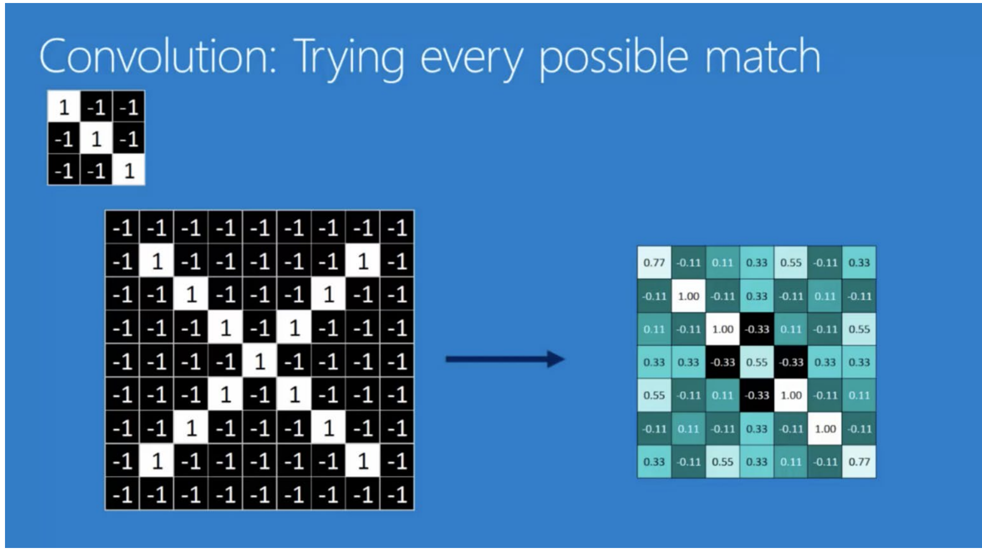
CNN Key Concepts and Architecture-- Convolution

A convolution is an operation that changes a function into something else. We do convolutions so that we can transform the original function into a form to get more information. Convolutions have been used for a long time in image processing to blur and sharpen images, and perform other operations, such as, enhance edges and emboss.

The convolution layer uses filters that perform convolution operations as it is scanning the input with respect to its dimensions. Its hyperparameters include the filter size and stride. The resulting output is called feature map or activation map.

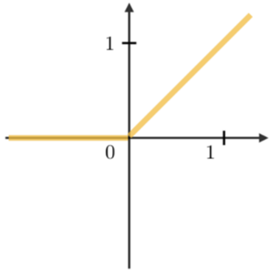
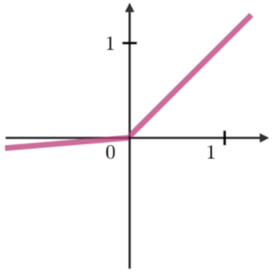
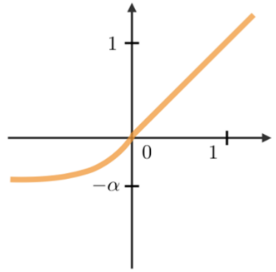






CNN Key Concepts and Architecture-- Commonly Used Activation Functions

❒ **Rectified Linear Unit** — The rectified linear unit layer (ReLU) is an activation function g that is used on all elements of the volume. It aims at introducing non-linearities to the network. Its variants are summarized in the table below:

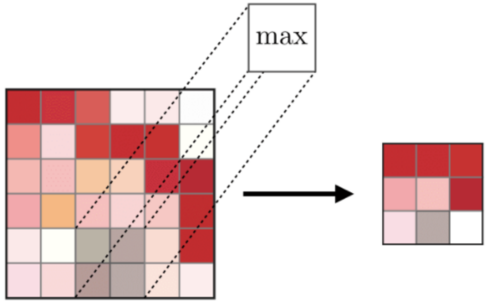
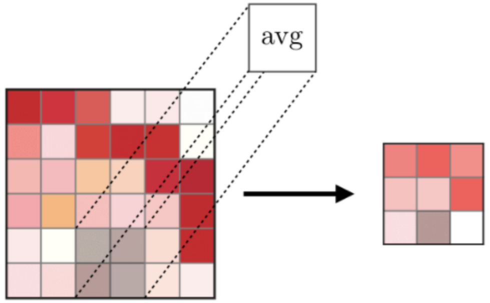
ReLU	Leaky ReLU	ELU
$g(z)=\max(0,z)$	$g(z)=\max(\epsilon z,z)$ with $\epsilon\ll1$	$g(z)=\max(\alpha(e^z-1),z)$ with $\alpha\ll1$
		
• Non-linearity complexities biologically interpretable	• Addresses dying ReLU issue for negative values	• Differentiable everywhere

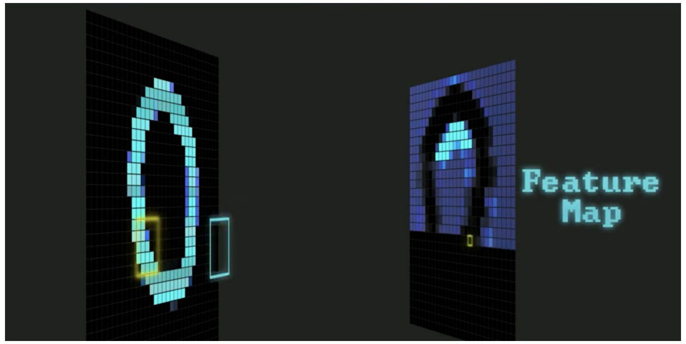
❒ **Softmax** — The softmax step can be seen as a generalized logistic function that takes as input a vector of scores $x\in\mathbb{R}^n$ and outputs a vector of output probability $p\in\mathbb{R}^n$ through a softmax function at the end of the architecture. It is defined as follows:

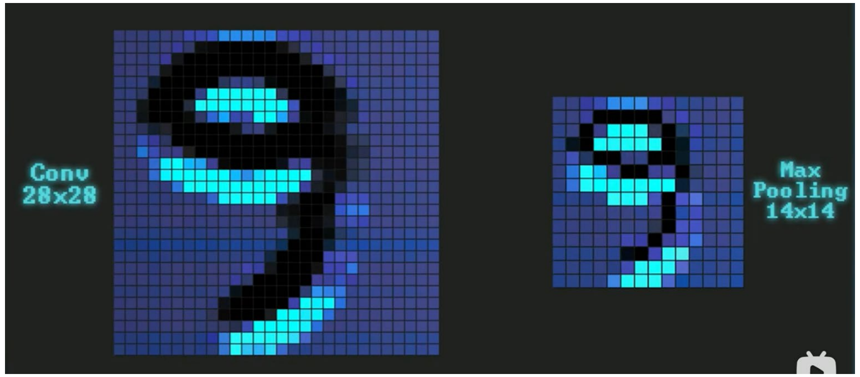
$$\boxed{p=\begin{pmatrix}p_1\\ \vdots\\ p_n\end{pmatrix}}\quad\text{where}\quad\boxed{p_i=\frac{e^{x_i}}{\displaystyle\sum_{j=1}^{n}e^{x_j}}}$$

CNN Key Concepts and Architecture -- Pooling

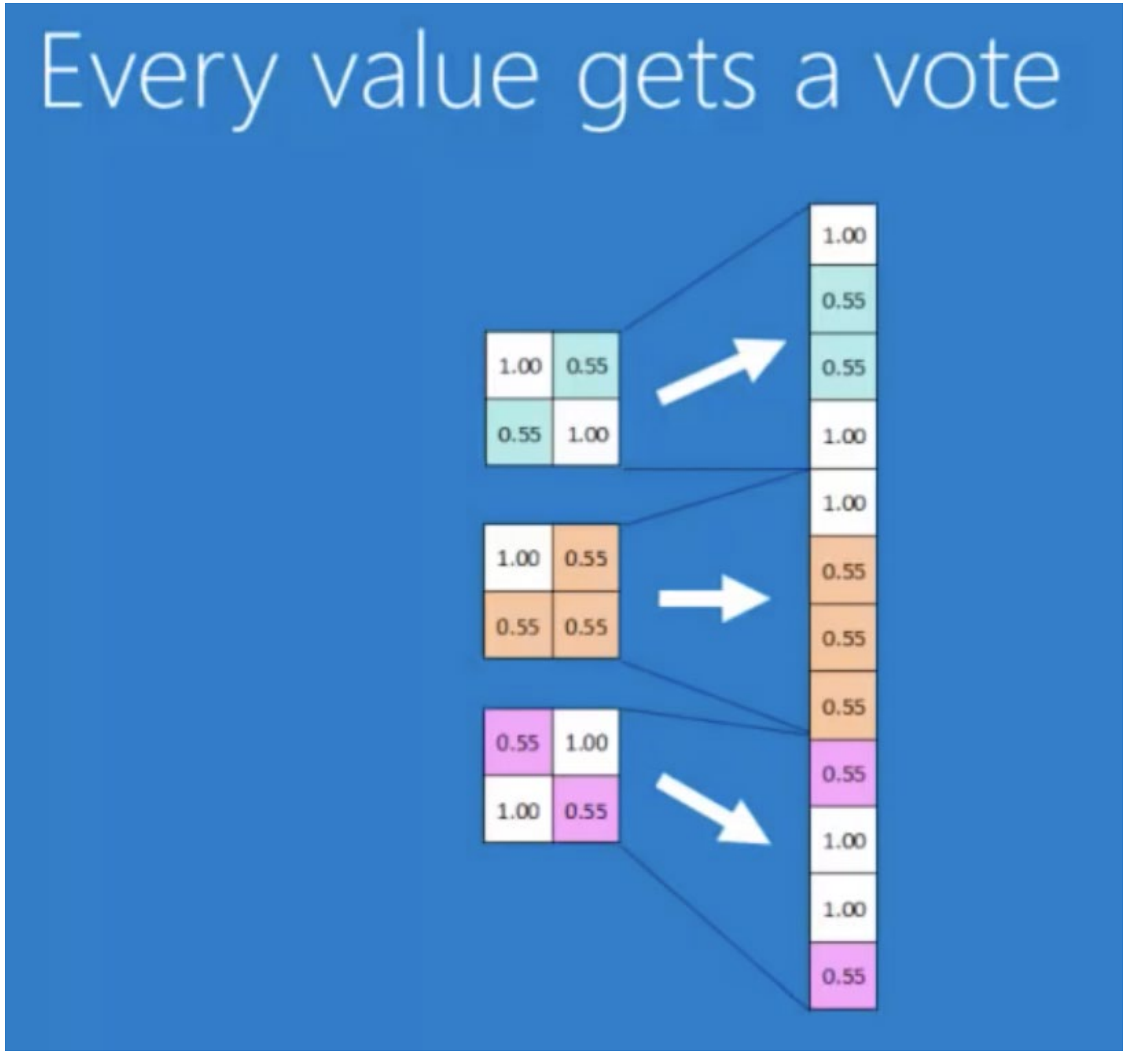
The pooling layer is a downsampling operation, typically applied after a convolution layer, which does some spatial invariance. In particular, max and average pooling are special kinds of pooling where the maximum and average value is taken, respectively.

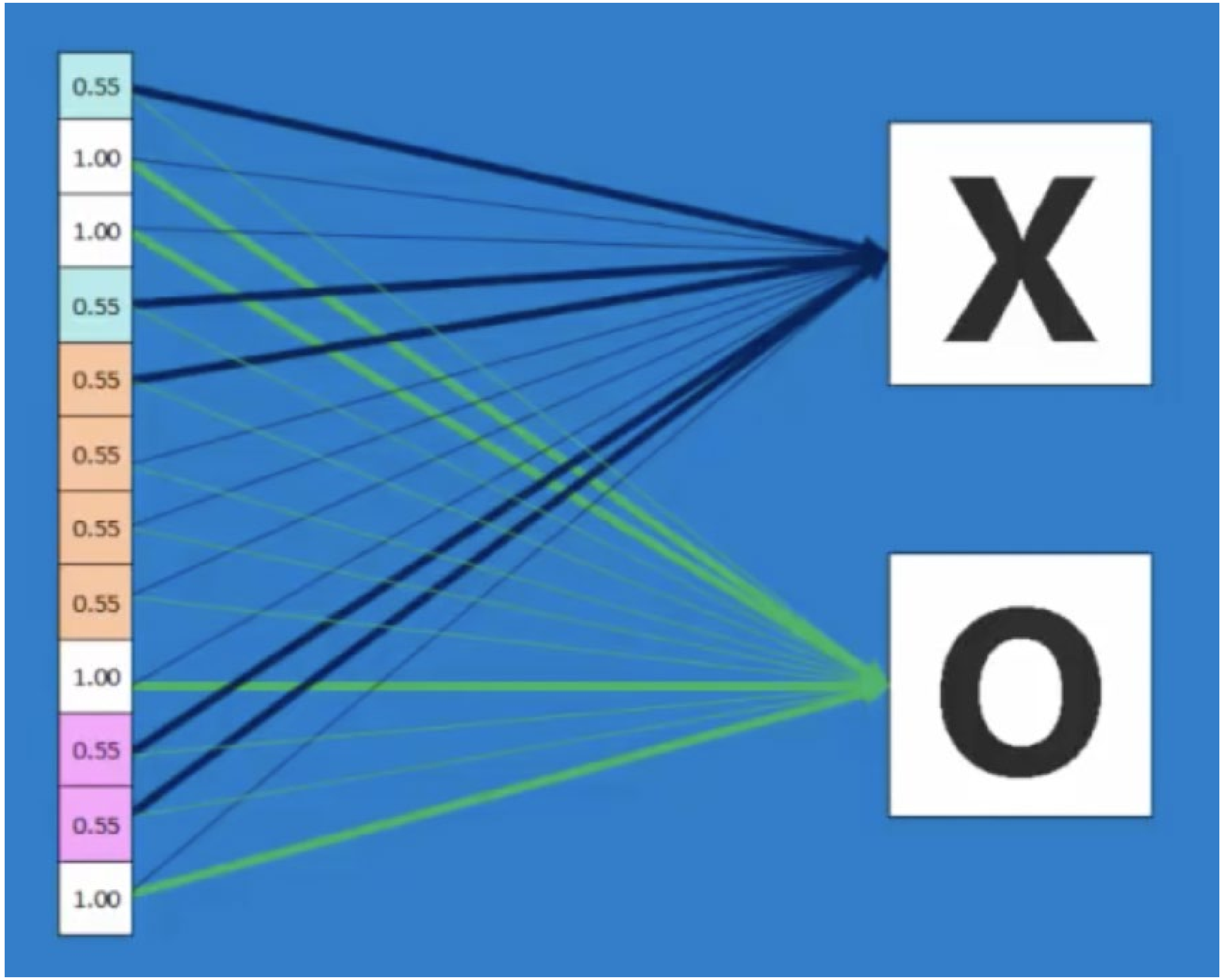
Type	Max pooling	Average pooling
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
Illustration		
Comments	• Preserves detected features • Most commonly used	• Downsamples feature map • Used in LeNet



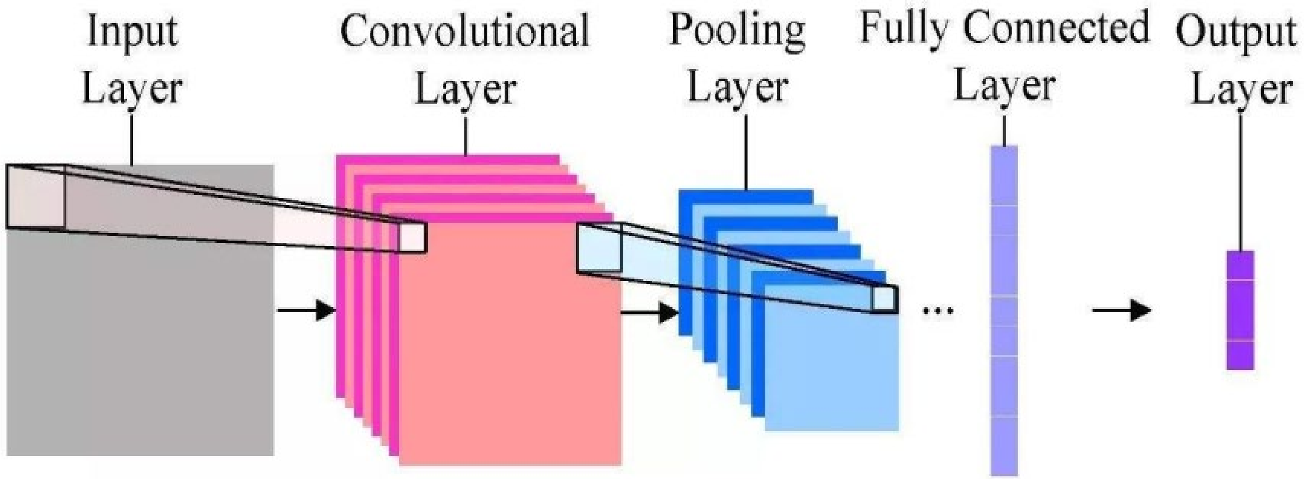


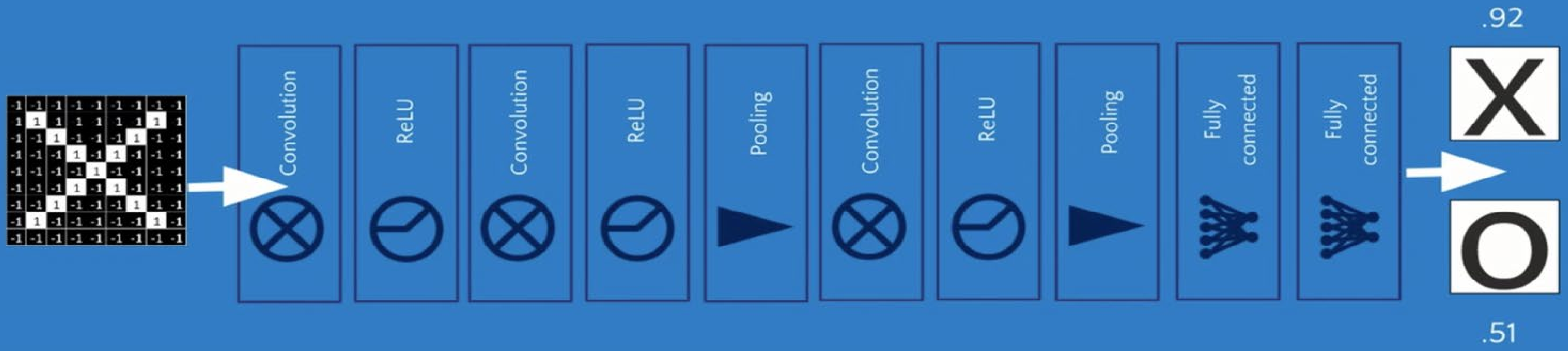
CNN Key Concepts and Architecture -- Fully Connected





Convolutional Neural Network Architecture

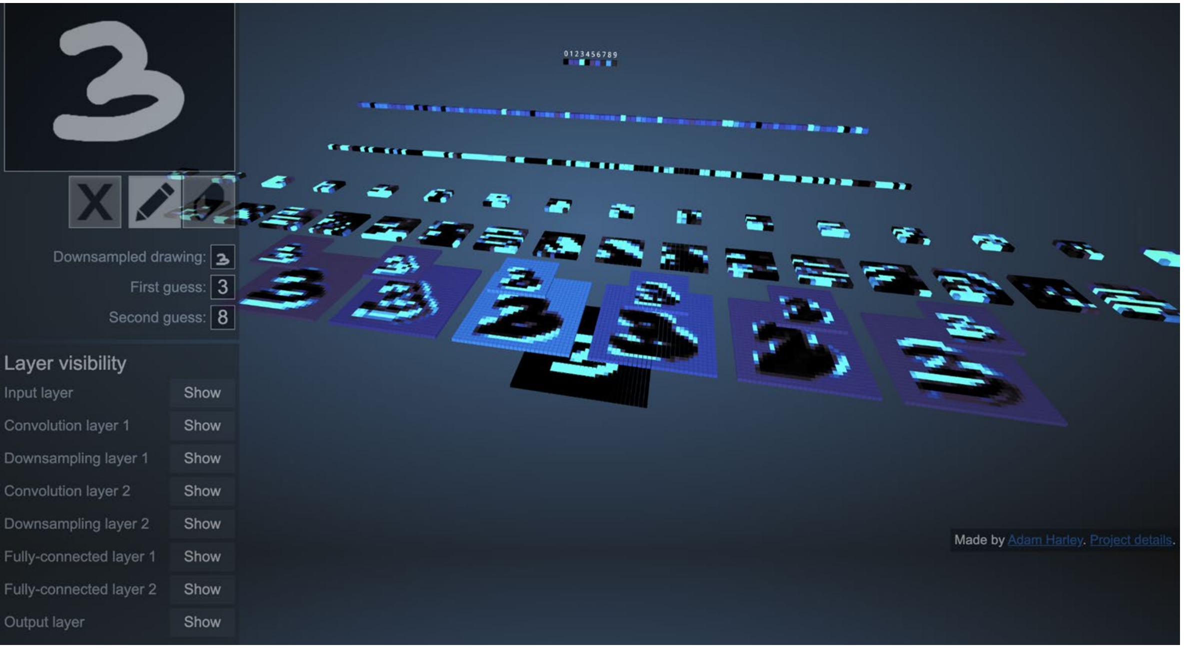


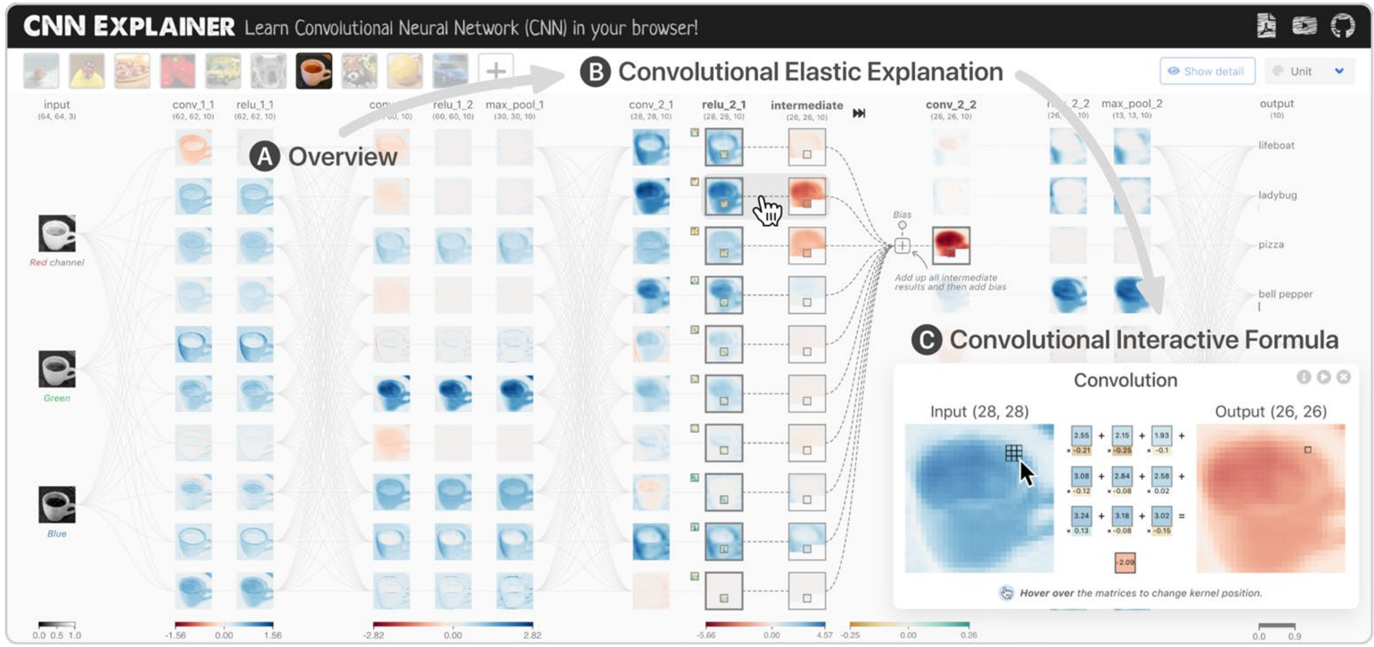


Visualization of Convolutional Neural Networks
(Adam Harley & Zijie Wang)

https://www.cs.ryerson.ca/~aharley/vis/conv/

https://zijie.wang/papers/cnn-explainer/

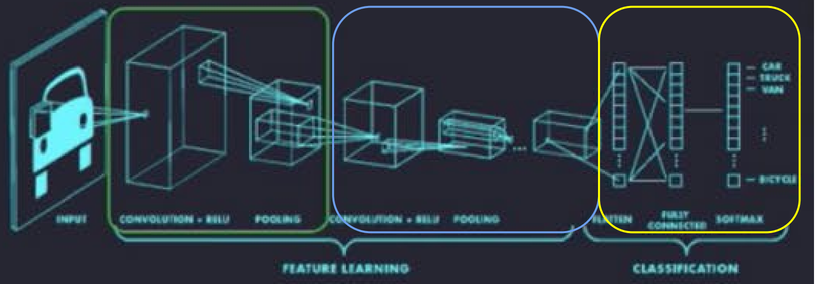




Convolutional Neural Network(CNN)

Concept to code

```
import tensorflow as tf

def generate_model():
    model = tf.keras.Sequential([
      # first convolutional layer
      tf.keras.layers.Conv2D(32, filter_size=3, activation='relu’),
      tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

      # second convolutional layer
      tf.keras.layers.Conv2D(64, filter_size=3, activation='relu’),
      tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

      # fully connected classifier
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(1024, activation='relu’),
      tf.keras.layers.Dense(10, activation=‘softmax’)   # 10 outputs
    ])
    return model
```



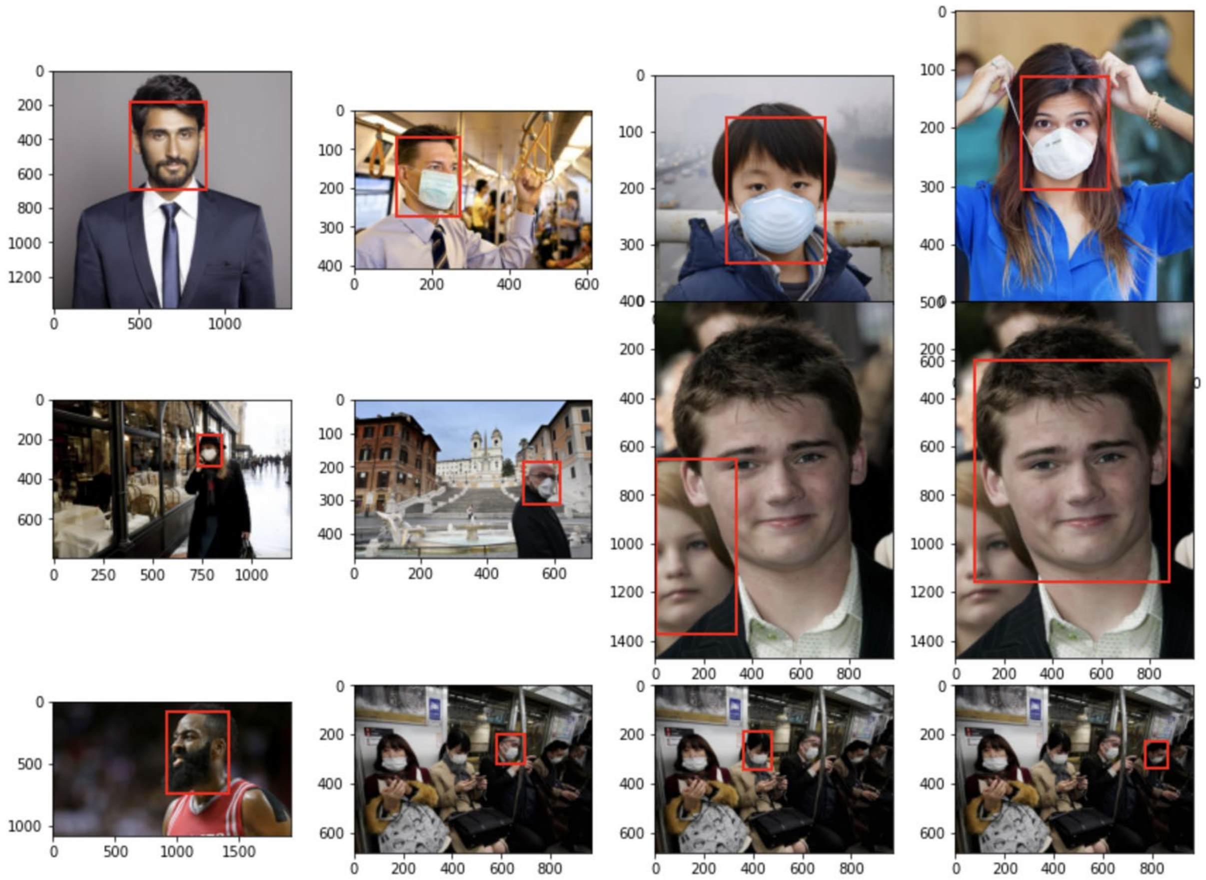
Part 4 Data Processing and Training

Data Processing

```
df = meta_data[(meta_data.classname=='face_with_mask') | (meta_data.classname=='face_no_mask')]
```

```
df.head()
```

	name	x1	x2	y1	y2	classname
0	1801.jpg	451	186	895	697	face_no_mask
1	1802.jpg	110	71	273	272	face_with_mask
2	1803.jpg	126	75	303	333	face_with_mask
3	1804.jpg	112	113	262	307	face_with_mask
4	1805.jpg	728	180	853	336	face_with_mask



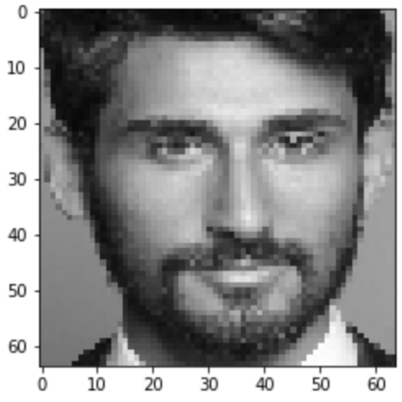
Total images: 3390

Total faces: 5749

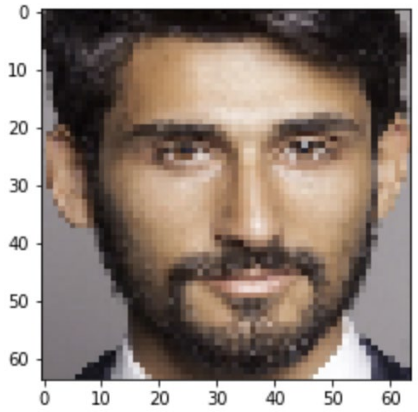
Data Processing

```
[17] def img_to_cv2(path, x1, x2, y1, y2, img_rows, img_cols, color_type=1):
         img_arr = []
         # Load as grayscale
         if color_type == 1:
             img_arr = cv2.imread(path, 0)
         elif color_type == 3:
             img_arr = cv2.imread(path)

         # extract face image
         img_arr = img_arr[x2:y2, x1:y1]
         # Reshape size
         resized = cv2.resize(img_arr, (img_cols, img_rows))

         return resized
```

Grayscale (64, 64, 1)



RGB(64, 64, 3)




```
X[0]
```

```
array([[100,  41,  20, ..., 139, 160, 160],
       [ 59,  44,  25, ...,  40, 160, 159],
       [ 63,  19,  28, ...,  29, 140, 168],
       ...,
       [118, 117, 118, ...,  35, 144, 150],
       [117, 117, 118, ...,  29,  45,  39],
       [116, 116, 115, ...,  28,  33,  48]], dtype=uint8)
```

1. **Read Image**
    a. The output of cv2.imread() is an array of BGR(Blue, Green, Red) values. (Use cv2.cvtColor to convert BGR to RGB for plotting)
    b. Use grayscale as an input.
2. **Extract face**
3. **Reshape face as input size**

Build model

Three convolution layers

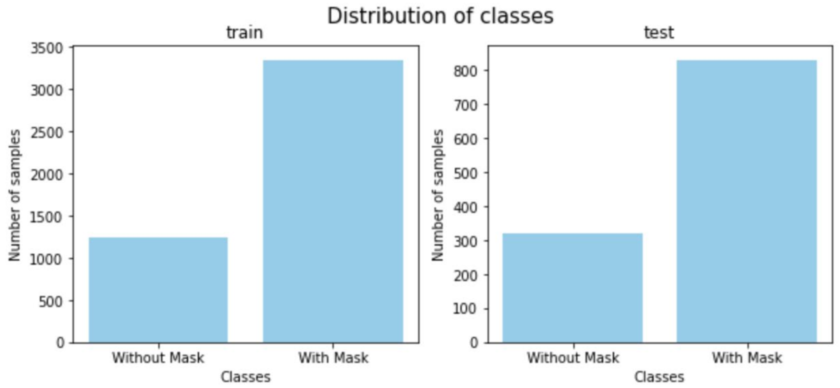
```
def create_cnn_model(img_rows, img_cols, color_type=1):
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_rows, img_cols, color_type)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))

    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))

    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))

    model.add(Flatten())
    model.add(Dense(2, activation='softmax'))

    model.summary()

    return model
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 62, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 31, 31, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 29, 29, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 2)                 9218
=================================================================
Total params: 101,890
Trainable params: 101,890
Non-trainable params: 0
```

Train model

- Normalize & split dataset

```
def normalize_data(X, y, img_rows, img_cols, color_type=1):
  X = np.array(X).reshape(-1, img_rows, img_cols, color_type)
  y = np.array(y)
  y = tf.keras.utils.to_categorical(y)
  X = X.astype('float32') / 255

  return X, y
```



- Set checkpoint to save the best model

```
[ ] def train_model(model, X_train, y_train, epochs, batch_size, checkpoint=None):
      model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])
      history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, validation_split=0.2,shuffle=True, callbacks=[checkpoint])
      return history
```

```
[ ] checkpoint_path = '/checkpoint'
    checkpoint = ModelCheckpoint(checkpoint_path, monitor='val_loss',verbose=0,save_best_only=True,mode='auto')
```

Train model

```
[5] # parameter
    img_rows, img_cols = 64, 64
    color_type = 1
    random_state = 45
    epochs = 20
    batch_size = 32
```

- train model

```
model = create_cnn_model(img_rows, img_cols, color_type)
history = train_model(model, X_train, y_train, epochs, batch_size, checkpoint)
```

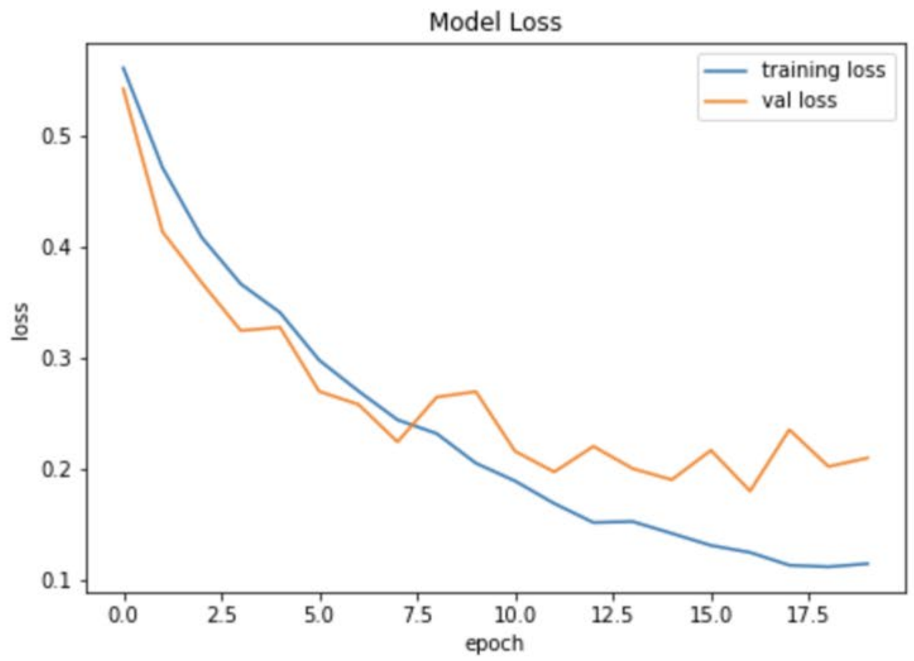
- load checkpoint to model

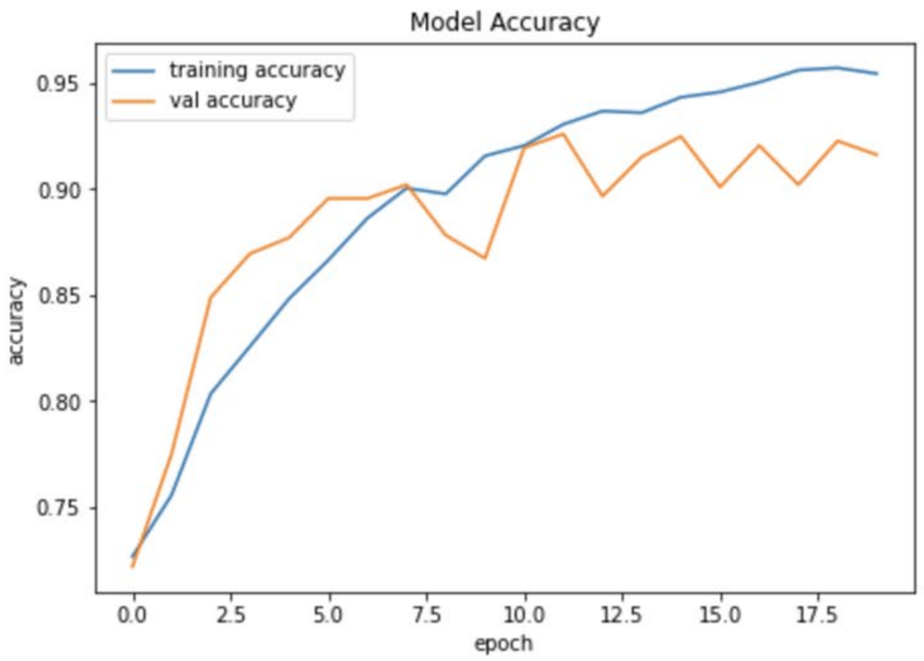
```
model.load_weights(checkpoint_path)
```

```
Epoch 1/20
115/115 [==============================] - 46s 247ms/step - loss: 0.5732 - accuracy: 0.7223 - val_loss: 0.5413 - val_accuracy: 0.7217
INFO:tensorflow:Assets written to: /checkpoint/assets
Epoch 2/20
115/115 [==============================] - 28s 240ms/step - loss: 0.4921 - accuracy: 0.7390 - val_loss: 0.4128 - val_accuracy: 0.7750
INFO:tensorflow:Assets written to: /checkpoint/assets
Epoch 3/20
115/115 [==============================] - 28s 241ms/step - loss: 0.4286 - accuracy: 0.7835 - val_loss: 0.3670 - val_accuracy: 0.8489
INFO:tensorflow:Assets written to: /checkpoint/assets
Epoch 4/20
115/115 [==============================] - 28s 241ms/step - loss: 0.3607 - accuracy: 0.8298 - val_loss: 0.3238 - val_accuracy: 0.8696
INFO:tensorflow:Assets written to: /checkpoint/assets
Epoch 5/20
115/115 [==============================] - 28s 245ms/step - loss: 0.3324 - accuracy: 0.8527 - val_loss: 0.3269 - val_accuracy: 0.8772
Epoch 6/20
115/115 [==============================] - 28s 246ms/step - loss: 0.3216 - accuracy: 0.8517 - val_loss: 0.2691 - val_accuracy: 0.8957
INFO:tensorflow:Assets written to: /checkpoint/assets
Epoch 7/20
115/115 [==============================] - 28s 246ms/step - loss: 0.2778 - accuracy: 0.8813 - val_loss: 0.2575 - val_accuracy: 0.8957
INFO:tensorflow:Assets written to: /checkpoint/assets
Epoch 8/20
115/115 [==============================] - 28s 243ms/step - loss: 0.2376 - accuracy: 0.9033 - val_loss: 0.2238 - val_accuracy: 0.9022
INFO:tensorflow:Assets written to: /checkpoint/assets
Epoch 9/20
115/115 [==============================] - 28s 244ms/step - loss: 0.2219 - accuracy: 0.8987 - val_loss: 0.2640 - val_accuracy: 0.8783
Epoch 10/20
115/115 [==============================] - 28s 243ms/step - loss: 0.2170 - accuracy: 0.9100 - val_loss: 0.2690 - val_accuracy: 0.8674
Epoch 11/20
115/115 [==============================] - 28s 242ms/step - loss: 0.2030 - accuracy: 0.9116 - val_loss: 0.2153 - val_accuracy: 0.9196
INFO:tensorflow:Assets written to: /checkpoint/assets
Epoch 12/20
115/115 [==============================] - 28s 241ms/step - loss: 0.1713 - accuracy: 0.9292 - val_loss: 0.1967 - val_accuracy: 0.9261
INFO:tensorflow:Assets written to: /checkpoint/assets
Epoch 13/20
115/115 [==============================] - 28s 243ms/step - loss: 0.1574 - accuracy: 0.9311 - val_loss: 0.2198 - val_accuracy: 0.8967
Epoch 14/20
115/115 [==============================] - 28s 242ms/step - loss: 0.1616 - accuracy: 0.9318 - val_loss: 0.1998 - val_accuracy: 0.9152
Epoch 15/20
115/115 [==============================] - 28s 243ms/step - loss: 0.1510 - accuracy: 0.9432 - val_loss: 0.1897 - val_accuracy: 0.9250
INFO:tensorflow:Assets written to: /checkpoint/assets
Epoch 16/20
115/115 [==============================] - 28s 243ms/step - loss: 0.1222 - accuracy: 0.9517 - val_loss: 0.2162 - val_accuracy: 0.9011
Epoch 17/20
115/115 [==============================] - 28s 243ms/step - loss: 0.1245 - accuracy: 0.9537 - val_loss: 0.1795 - val_accuracy: 0.9207
INFO:tensorflow:Assets written to: /checkpoint/assets
Epoch 18/20
115/115 [==============================] - 28s 243ms/step - loss: 0.1149 - accuracy: 0.9578 - val_loss: 0.2347 - val_accuracy: 0.9022
Epoch 19/20
115/115 [==============================] - 28s 245ms/step - loss: 0.1225 - accuracy: 0.9507 - val_loss: 0.2016 - val_accuracy: 0.9228
Epoch 20/20
115/115 [==============================] - 28s 244ms/step - loss: 0.1152 - accuracy: 0.9536 - val_loss: 0.2093 - val_accuracy: 0.9163
```

Part 5 Testing and Evaluation

Visualize training and evaluate testing set

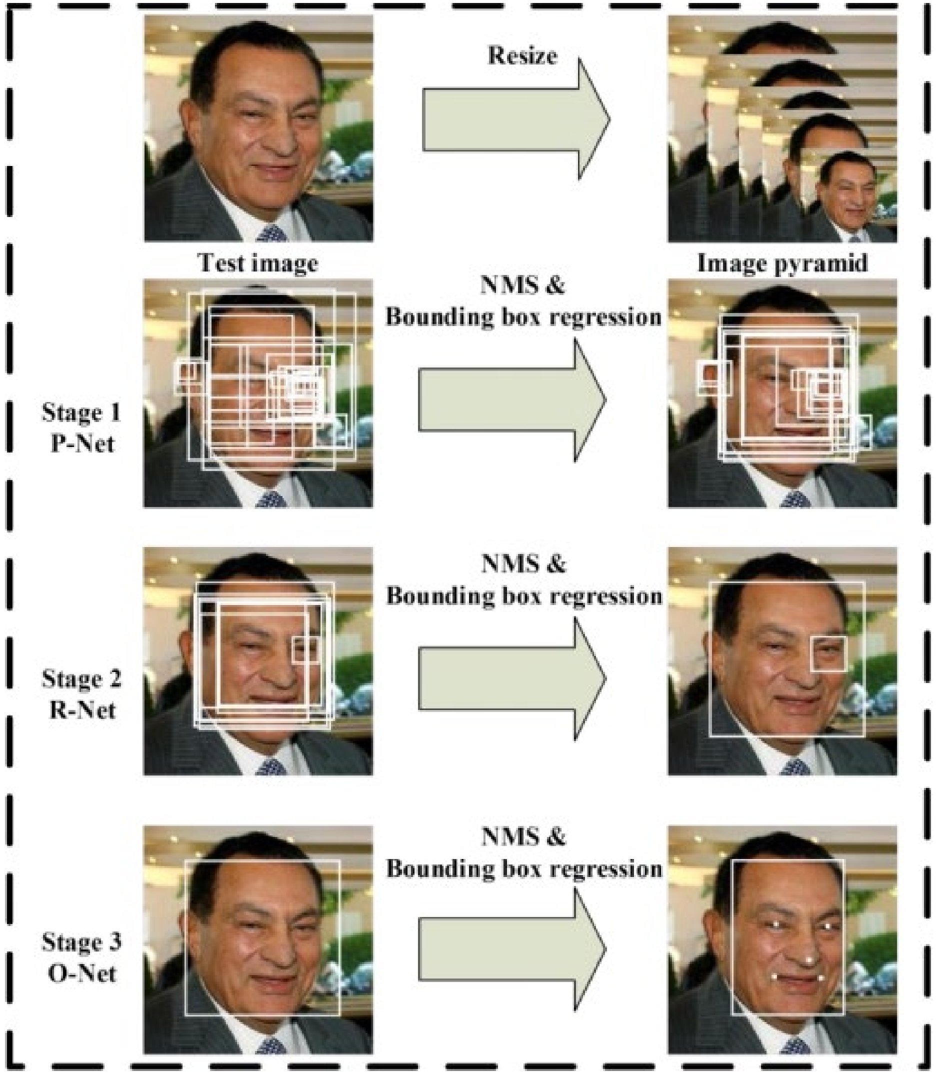





```
train_score = model.evaluate(X_train, y_train, batch_size=batch_size)
test_score = model.evaluate(X_test, y_test, batch_size=batch_size)
```

```
144/144 [==============================] - 9s 59ms/step - loss: 0.1024 - accuracy: 0.9630
36/36 [==============================] - 2s 59ms/step - loss: 0.1941 - accuracy: 0.9252
```

Single image prediction - MTCNN

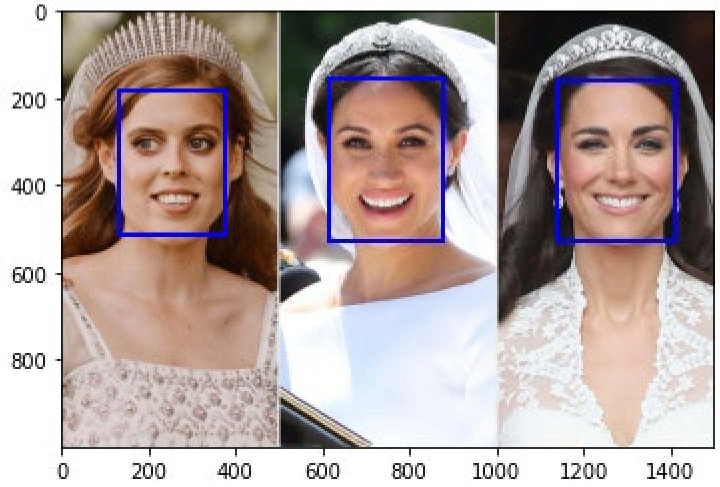


First Stage: P-Net(Proposal Net). Create multiple frames scan through the whole image.

Second Stage: R-Net(Refinement Net). Use information from P-Net as input, frames which do not contain faces will be rejected by the R-Net.

Third Stage: O-Net(Output Net). O-Net will eventually outputs the face and facial landmarks position detecting from the image.

Single image prediction




```python
def face_detect(path):
    box_arr = []
    img = plt.imread(path)
    faces = MTCNN().detect_faces(img)
    for face in faces:
        box_arr.append(face['box'])
    return box_arr
```

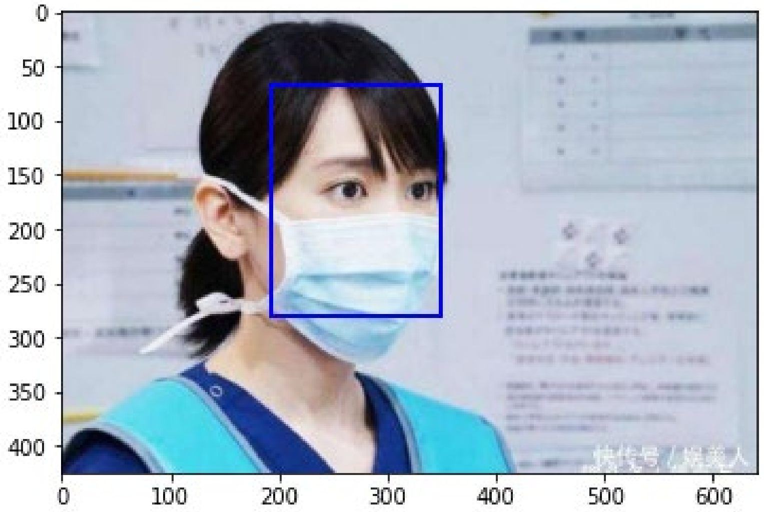
```python
file_name = "test.jpeg"
path = file_name

fig,ax = plt.subplots()
img = plt.imread(path)
ax.imshow(img)

# x1, y1, width, height = face['box']
faces = face_detect(path)
for i,face in enumerate(faces):
    x1, y1, width, height = face[0], face[1], face[2], face[3]
    face = patches.Rectangle((x1,y1),width,height,linewidth=2,edgecolor='b',facecolor='none')
    ax.add_patch(face)

plt.show()
```

```python
data_1 = load_test_data(path, img_rows, img_cols, color_type)
data_1 = normalize_test_data(data_1, img_rows, img_cols, color_type)
```

```python
model.predict(data_1)
```

```
array([[9.8476613e-01, 1.5233894e-02],
       [9.9991572e-01, 8.4258529e-05],
       [9.9990559e-01, 9.4352967e-05]], dtype=float32)
```

Single image prediction




```
file_name = "test2.jpeg"
path = file_name

fig,ax = plt.subplots()
img = plt.imread(path)
ax.imshow(img)

# x1, y1, width, height = face['box']
faces = face_detect(path)
for i,face in enumerate(faces):
        x1, y1, width, height = face[0], face[1], face[2], face[3]
        face = patches.Rectangle((x1,y1),width,height,linewidth=2,edgecolor='b',facecolor='none')
        ax.add_patch(face)

plt.show()
```

```
data_2 = load_test_data(path, img_rows, img_cols, color_type)
data_2 = normalize_test_data(data_2, img_rows, img_cols, color_type)
```

```
model.predict(data_2)
```

```
array([[0.01019729, 0.9898028 ]], dtype=float32)
```

Part 6 Conclusion

Thank you

References

How Convolutional Neural Networks Work (CNNs Explained & Visualized)
https://www.youtube.com/watch?v=pj9-rr1wDhM

How Convolutional Neural Networks work
https://www.youtube.com/watch?v=FmpDIaiMIeA

MIT 6.S191 (2020): Convolutional Neural Networks
https://www.youtube.com/watch?v=iaSUYvmCekI

Convolutional Neural Networks cheatsheet
https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

An Interactive Node-Link Visualization of Convolutional Neural Networks
https://www.cs.ryerson.ca/~aharley/vis/conv/

Different Kinds of Convolutional Filters
https://www.saama.com/different-kinds-convolutional-filters/

How convolutional neural networks work, in depth
https://www.youtube.com/watch?v=JB8T_zN7ZC0

CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization
https://zijie.wang/papers/cnn-explainer/

https://www.kaggle.com/dabawse/detecting-face-masks-with-5-models/data