https://cn.overleaf.com/project/64770d24101975e70940bb67

OCR for Chinese Characters with CNN and EfficientNet

Yubo Cai and Junyuan Wang

CSE204 Machine Learning, Ecole Polytechnique

2022-2023

Why OCR in Chinese?

Reasons for application requirements:

- Important and widely used language
- A large amount of Chinese historical research material needs to be electronic

Personal Interests:

- Intuitive ideas for recognizing letters and numbers from MNIST datasets in your own native language.
- More interesting and challenging.

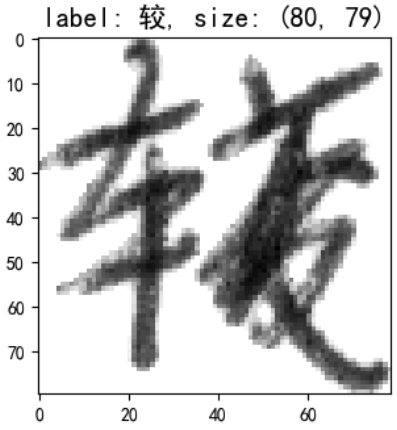
Difficulties compared to MNIST

	Chinese Characters	MNIST (English Alphabet)
Number of Characters	10k+	26
Structure	More Complex	Relatively simple
Diversity of writing styles	A large number of writing styles	More fixed

Table: Comparison between Chinese Characters and MNIST (English Alphabet)

楷楷
書书

Table: Writing style: Kaishu and Cursive



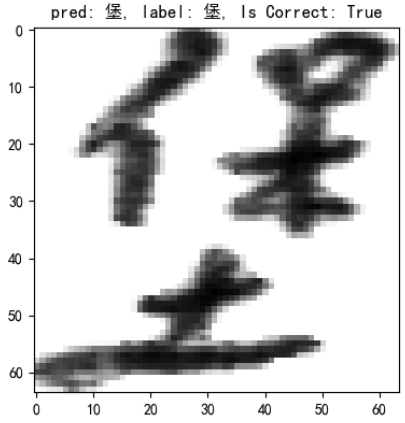


Table: Top-bottom and left-right structures

- Large Amount of characters need to label
- Multiple hand-writing styles
- Variety of font structures

Data Description & Preprocessing

Datasets in the Project: **CASIA Online and Offline Chinese Handwriting Databases**. Here we use **CASIA-HWDB1.0-1.2**. Key information of the datasets:

- 7185 Chinese characters
- 171 English letters, numbers
- 897758 pictures in the training dataset
- 223991 pictures in the testing dataset
- Single character images

Figure: Exmaple pictures of the dataset

Data Description & Preprocessing

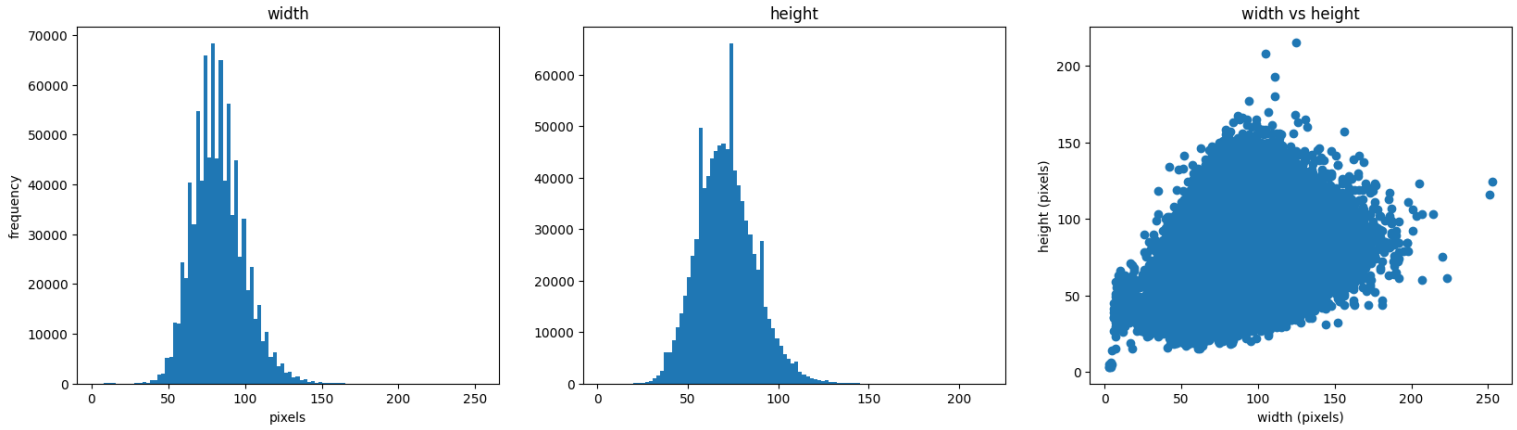


Figure: Plots of the training datasets

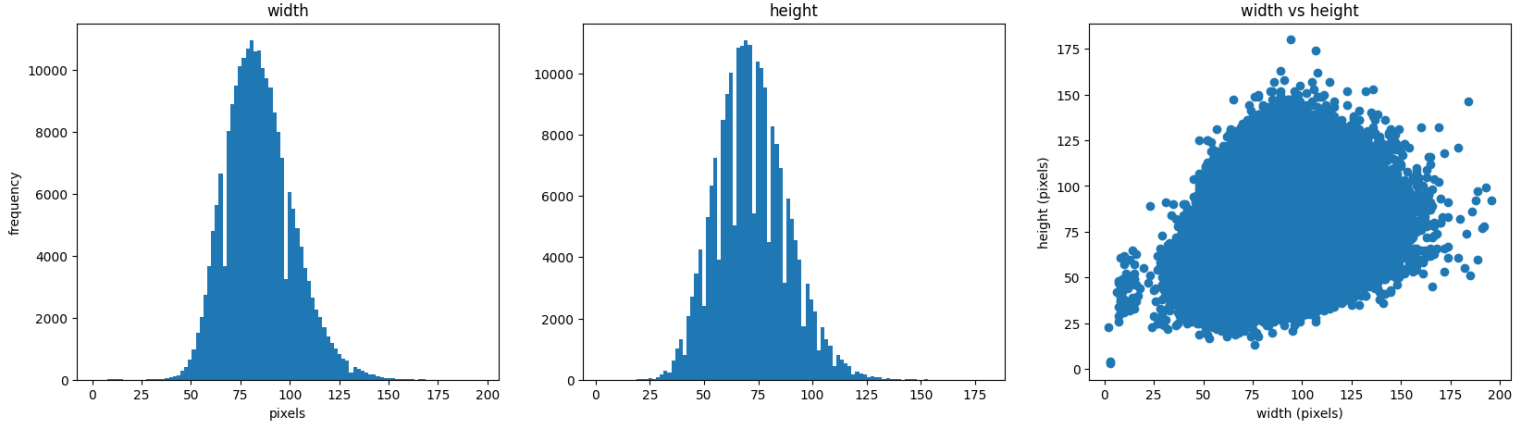


Figure: Plots of the testing datasets

Data Description & Preprocessing

Data Preprocessing: From **gnt** file to **tfrecord**:

- Label: decode with **GB2312** to Chinese, then **tf.cast()** transfer into **tf.int64**.
- Image: Using **tf.io.decode_raw()** decode image data into **tf.uint8**, apply **tf.reshape()** transfer from 1D to 2D.

More detailed information on *convert_to_tfrecord.py*

Image Processing:

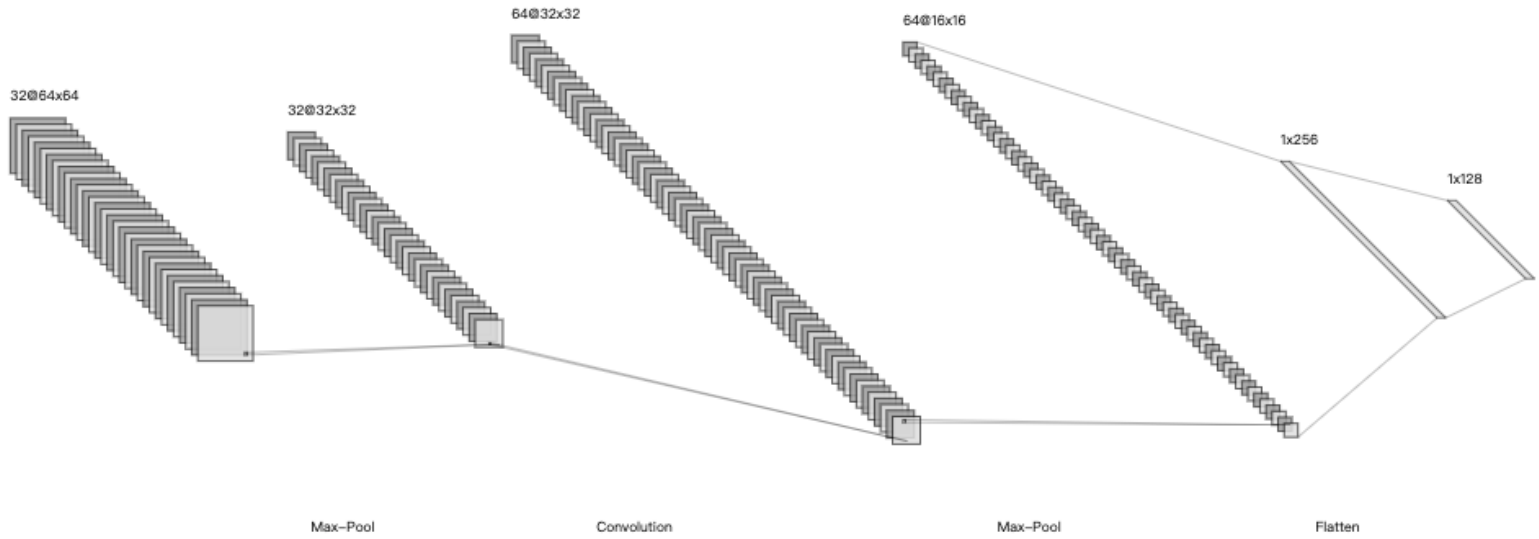
- Segmentation
- Reshaping to 64×64 for CNN
- Reshaping to 224×224 for EfficientNet B0
- Background uniformization
- Edge finding (for segmentation)
- Image grayscale adjustment

Simple CNN

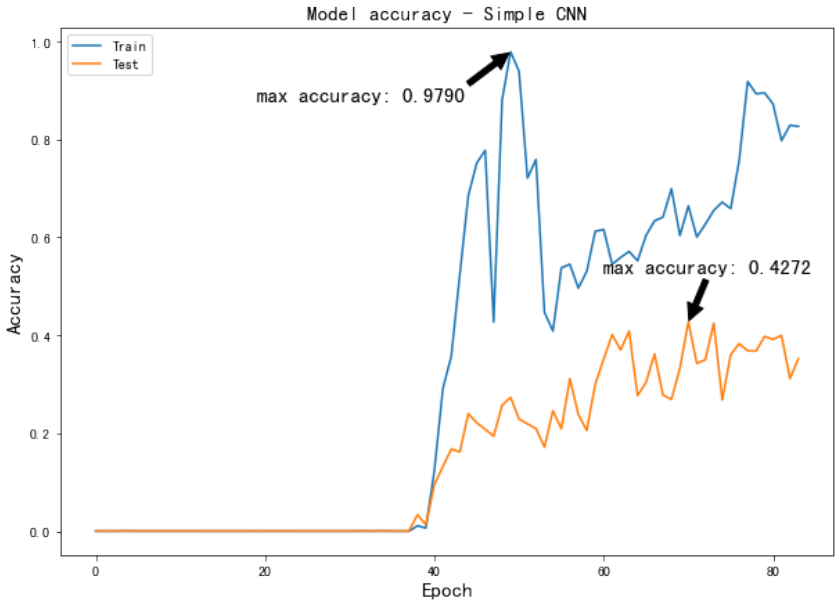
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 32)        320

 max_pooling2d (MaxPooling2D  (None, 32, 32, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 32, 32, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 flatten (Flatten)           (None, 16384)             0

 dense (Dense)               (None, 3755)              61525675

=================================================================
Total params: 61,544,491
Trainable params: 61,544,491
Non-trainable params: 0
_________________________________________________________________
```

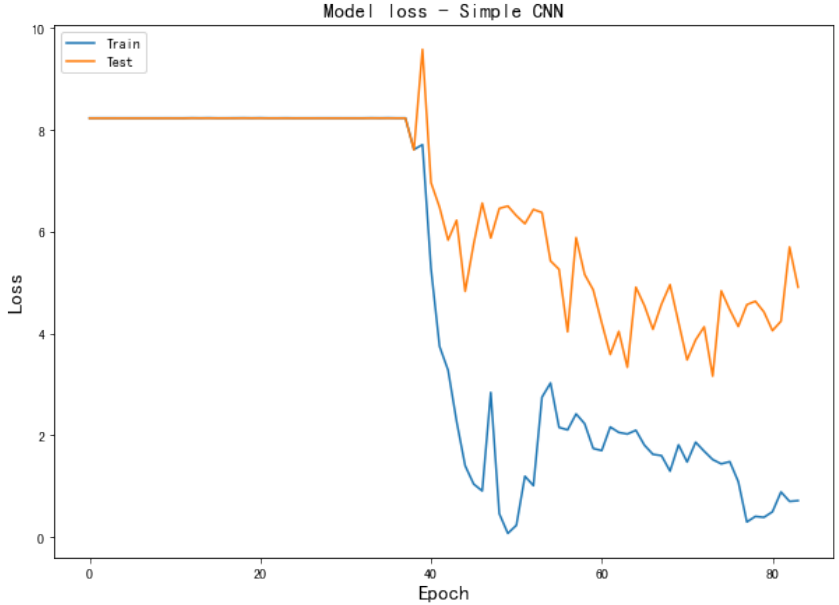
Since the dataset is quite large and the model might converge very slow, we use the checkpoint saving and logging callback functions to tracking the training of model.

```
tf.keras.callbacks.ModelCheckpoint(ckpt_path,
                                   save_weights_only=True,
                                   verbose=1,
                                   save_freq='epoch',
                                   save_best_only=True),
```



Result of Simple CNN





Accuracy on training dataset: around 95%. Accuracy on the testing dataset: around 46% after 120 epochs.

Result of Simple CNN

The result is good for such a simple CNN model to recognize complex Chinese characters, but how can we improve the accuracy?

Figure: Random Choice of 9 pictures from the testing dataset, 4 are correct

Complex CNN Model

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 64, 64, 64)        640

max_pooling2d_2 (MaxPooling  (None, 32, 32, 64)        0
2D)

conv2d_3 (Conv2D)            (None, 32, 32, 128)       73856

max_pooling2d_3 (MaxPooling  (None, 16, 16, 128)       0
2D)

conv2d_4 (Conv2D)            (None, 16, 16, 256)       295168

max_pooling2d_4 (MaxPooling  (None, 8, 8, 256)         0
2D)

conv2d_5 (Conv2D)            (None, 8, 8, 512)         1180160

max_pooling2d_5 (MaxPooling  (None, 4, 4, 512)         0
2D)

flatten_1 (Flatten)          (None, 8192)              0

dense_1 (Dense)              (None, 1024)              8389632

dense_2 (Dense)              (None, 3755)              3848875

=================================================================
Total params: 13,788,331
Trainable params: 13,788,331
Non-trainable params: 0
_________________________________________________________________
```

```python
# Define with a more complex CNN model - However, this model is not even converging
def CNN_ComplexModel_1(input_shape, n_classes):
    model = tf.keras.Sequential([
        layers.Conv2D(input_shape=input_shape, filters=64, kernel_size=(3, 3), strides=(1, 1),
                      padding='same', activation='relu'),
        layers.MaxPool2D(pool_size=(2, 2), padding='same'),
        layers.Conv2D(filters=128, kernel_size=(3, 3), padding='same'),
        layers.MaxPool2D(pool_size=(2, 2), padding='same'),
        layers.Conv2D(filters=256, kernel_size=(3, 3), padding='same'),
        layers.MaxPool2D(pool_size=(2, 2), padding='same'),
        layers.Conv2D(filters=512, kernel_size=(3, 3), padding='same'),
        layers.MaxPool2D(pool_size=(2, 2), padding='same'),

        layers.Flatten(),
        layers.Dense(1024, activation='relu'),
        layers.Dense(n_classes, activation='softmax')
    ])
    return model
```

(function) pool_size: Any

Result: To our surprise, the complex CNN model does not converge. But Why?

Result Analysis

Why Simple CNN works:

- We found that simple CNN has better feature extraction for Chinese characters with a clear structure in the dataset. For example, left-right structure, and up-down structure.

Why 95% accuracy in training and 40% in testing:

- **Limited feature extraction capability:** We found that simple CNNs are less capable of extracting features for more complex structured Chinese characters, especially for the problem of consecutive strokes due to writing style, which is limited by the small number of parameters.

Why Complex CNN not converge:

- **Optimization Difficulty:** [1] Deeper networks are harder to train. Gradients can vanish or explode as they are backpropagated through many layers, making learning difficult.
- **Not Suitable with dataset:** Complex CNN models are more demanding on the training dataset, and the training is negatively affected by the high level of noise in the training set and some labeling errors.

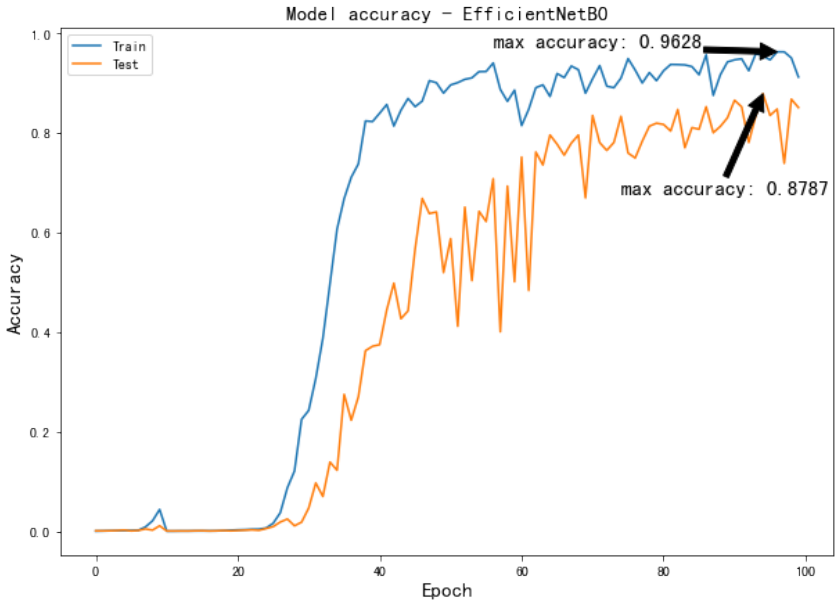
Model Building for EfficientNet B0

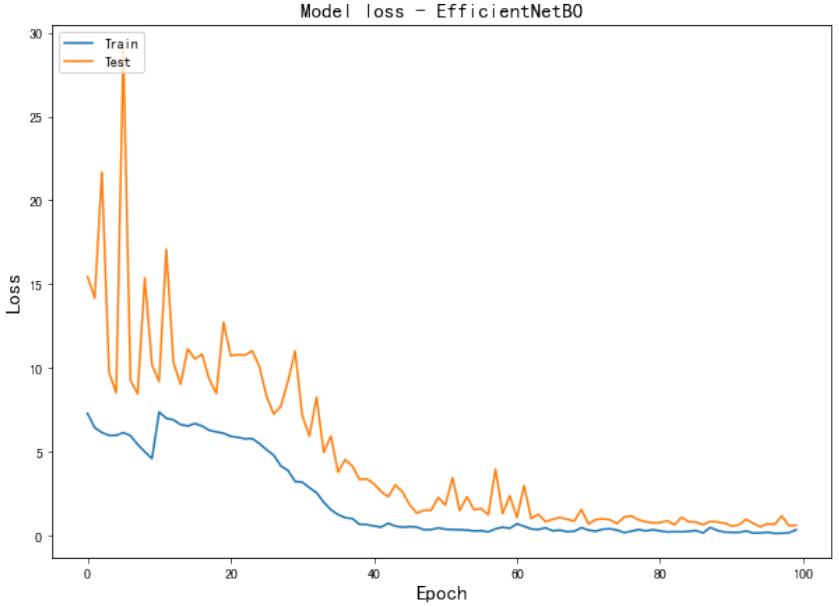
Image data processing: Resize the image to (224, 224). No normalization is needed for EfficientNetB0.

```
def effcientnetB0_model():
    # EfficientNetB0 expects 3 channels
    inputs = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
    base_model = EfficientNetB0(
        include_top=False, input_tensor=inputs, weights='imagenet')
    x = base_model.output
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(num_classes, activation='softmax')(x)
    model = keras.Model(inputs=inputs, outputs=x)
    return model
```

Here we use **tf.Keras** API to build the EfficientNet B0.

Result of EfficientNet





Accuracy on training dataset: around 96%. Accuracy on the testing dataset: around 87% after 80 epochs.

Result of EfficientNet

The EfficientNet model has greatly improved our accuracy. Then what is the advantage compared to CNN? And why such a complex network does not present the problems of the previous complex CNN models?

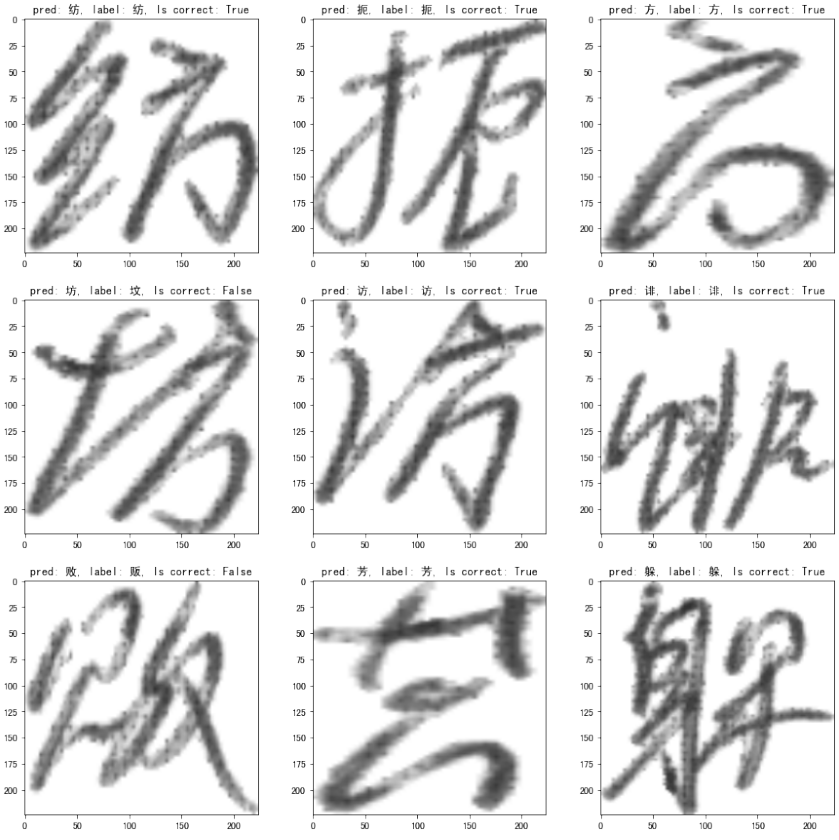


Figure: Random Choice of 9 pictures from the testing dataset, 7 are correct

Why EfficientNet Perform better?

Here we cite from the paper of EfficientNet [2]:

- **Increase the depth of Network**:
  - Advantages: Obtain richer, more complex features.
  - Problems: Gradient Vanish, Hard to train the model.
- **Increase the width of Network**:
  - Advantages: Obtain higher fine-grained features and be easy for training.
  - Problems: It is difficult to learn deep features for networks with shallow depth.
- **Increase image resolution**:
  - Advantages: Obtain higher fine-grained features.
  - Problems: Heavy computation.

Image data processing: The best ratio of **width, depth, and resolution** of the model is obtained through **NAS (Neural Architecture Search)** technology to achieve better results.

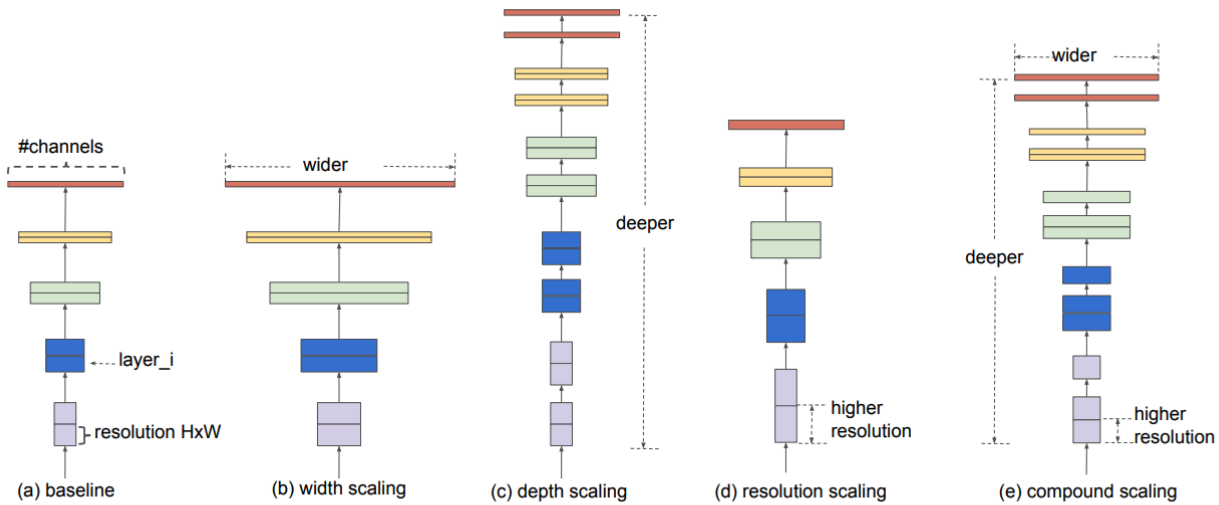


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

Go further?

A natural idea is we can extend this model to recognize whole paragraphs of text. We have two ideas to expand our project in this direction:

1. **Segmentation with edge detection.**
2. **CRNN (CNN+LSTM+CTC)** [3] [4]

Link of the GitHub Repository: [https://github.com/yubocai-poly/Chinese-character-OCR-with-CNN-and-EfficientNet]

Reference

[1] Ralf C Staudemeyer and Eric Rothstein Morris. "Understanding LSTM–a tutorial into long short-term memory recurrent neural networks". In: *arXiv preprint arXiv:1909.09586* (2019).

[2] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: 1905.11946 [cs.LG].

[3] Alex Graves et al. "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks". In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 369–376.

[4] Yong Yu et al. "A review of recurrent neural networks: LSTM cells and network architectures". In: *Neural computation* 31.7 (2019), pp. 1235–1270.