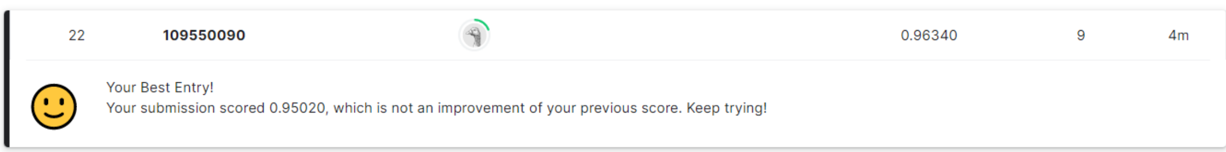
NYCU Pattern Recognition, Final Project

> 109550090 李以恩

Part. 1, Kaggle Submission (60%):

- Take a screenshot of your results on the public leaderboard and paste it on the report.
- Your score will be determined by the private leaderboard, but we will verify if your results match.
- The maximum number of entries allowed per day is 5.



Part. 2, Report (40%):

📌Environment details

1. **GPU**

```
root@baadc731898e:/# nvidia-smi
Mon Jun  5 07:38:21 2023
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 530.30.02              Driver Version: 530.30.02    CUDA Version: 12.1     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                  Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf            Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA GeForce GTX 1080         On | 00000000:01:00.0 Off |                  N/A |
|  0%   37C    P8               10W / 240W|    123MiB /  8192MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
```

2. **Docker container**

- Download docker image from 11.0.3-cudnn8-devel-ubuntu18.04
- Create a container

```
docker pull nvidia/cuda:11.0.3-devel-ubuntu18.04
docker run -d -it --gpus all -p [Port] -v [Path]:[Container Path] --name [name] [image ID]
```

3. **Execute the container and update**

- python3 —version: Python 3.6.9

```
apt-get update
apt-get upgrade -y
apt-get install python3 -y
apt-get install python3-pip -y
pip3 install --upgrade pip
```

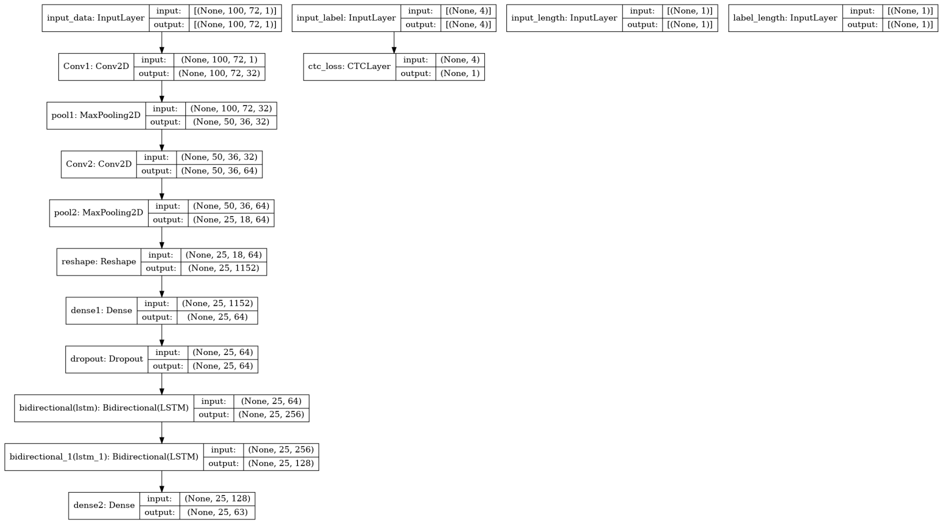
4. **Install some requirements module**

- See more in requirements.txt.

```
pip install tensorflow
pip install opencv-python
pip install scikit-learn
pip install matplotlib
pip install pandas
pip install pydot
```

📌Implementation details

1. **Model architecture**



2. **Hyperparameters**
    - train_epoch = 100
    - batch_size = 8
    - img_width = 100
    - img_height = 72
    - monitor = 'val_loss'
    - patience = 100
    - seed = 1

3. **Used deep learning framework**
    a. CNN: for feature extraction.
    b. RNN: for sequence recognition.
    c. CTC loss: for addressing the problem of label and output misalignment by removing consecutive duplicate characters and blank characters.

4. **Experimental design**

A detailed description of your experimental design, including the methodology and procedures employed in your study.

- **Characters processing**
  - Store all the possible characters in a set. Then create a dictionary **char_to_labels** to map the text to numeric labels

```
characters = set()                                      # Store all the characters in a set
captcha_length = []                                     # A list to store the length of each captcha
label_name = dataset["label"]
for word in label_name :
    captcha_length.append(len(word))
    for ch in word :
        characters.add(ch)
characters = sorted(characters)
char_to_labels = {char:idx for idx, char in enumerate(characters)}  # Map text to numeric labels
labels_to_char = {val:key for key, val in char_to_labels.items()}   # Map numeric labels to text
max_length = max(Counter(captcha_length).keys())

print(char_to_labels)
print(labels_to_char)
```

  - Record the max length of the label, and set to **max_length**. When preparing batches for the dataset in DataGenerator(), for images whose labels are less than max_length would be appended -1 to fullfill to max_length. Therefore, there is no need to customize the code for task1, task2 and task3.

```
def __getitem__(self, idx):
    curr_batch_idx = self.indices[idx*self.batch_size:(idx+1)*self.batch_size]
    batch_len = len(curr_batch_idx)

    batch_images = np.ones((batch_len, self.img_width, self.img_height, 1), dtype=np.float32)
    batch_labels = np.ones((batch_len, self.max_length), dtype=np.float32)
    input_length = np.ones((batch_len, 1), dtype=np.int64) * (self.img_width // self.downsample_factor - 2)
    label_length = np.zeros((batch_len, 1), dtype=np.int64)

    for j, idx in enumerate(curr_batch_idx):
        img = self.data[idx].T
        img = np.expand_dims(img, axis=-1)
        text = self.labels[idx]
        if is_valid_captcha(text, self.characters):
            label = [self.char_map[ch] for ch in text]
            for _ in range(self.max_length - len(label)):
                label.append(-1)
            batch_images[j] = img
            batch_labels[j] = label
            label_length[j] = len(text)

    batch_inputs = {
            'input_data': batch_images,
            'input_label': batch_labels,
            'input_length': input_length,
            'label_length': label_length,
            }
```

- **Data preprocessing**
  - Shuffle the data

```
dataset = pd.read_csv(f"{dataset_path}/train/annotations.csv")          # Store image-label info
dataset["filepath"] = f"{dataset_path}/train/" + dataset["filename"]    # Append "filepath"
dataset = dataset.sample(frac=1.).reset_index(drop=True)                # Shuffle the dataset
print(dataset.head())
```

  - Resized the images

```
for i in range(num_items):
    img = cv2.imread(df["filepath"][i])
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    if resize:
        img = cv2.resize(img, (img_width, img_height))
    img = (img/255.).astype(np.float32)
    label = df["label"][i]
```

    - Hyperparameters

      img_width = 100
      img_height = 72

    - Although training images are in the shape of squares, resizing images to 100x100 got worse performance. I guess it is because stretching the image into a wider shape makes the digits easier to recognize.

- **CTC loss**

  CTC (Connectionist Temporal Classification) loss is a loss function used for sequence classification tasks and the computation of CTC loss is based on the alignment between the label sequence and the predicted sequence by the model. CTC loss is suitable for problems with variable-length input and output sequences, such as speech recognition and **optical character recognition (OCR)**.

  The call method in the CTCLayer uses the **keras.backend.ctc_batch_cost** function to calculate the CTC loss and adds the computed loss value to the layer using the self.add_loss() method.

```python
class CTCLayer(layers.Layer):
    def __init__(self, name=None, **kwargs):
        super().__init__(name=name, **kwargs)
        self.loss_fn = keras.backend.ctc_batch_cost

    def call(self, y_true, y_pred, input_length, label_length):
        loss = self.loss_fn(y_true, y_pred, input_length, label_length)
        self.add_loss(loss)
        return loss
```

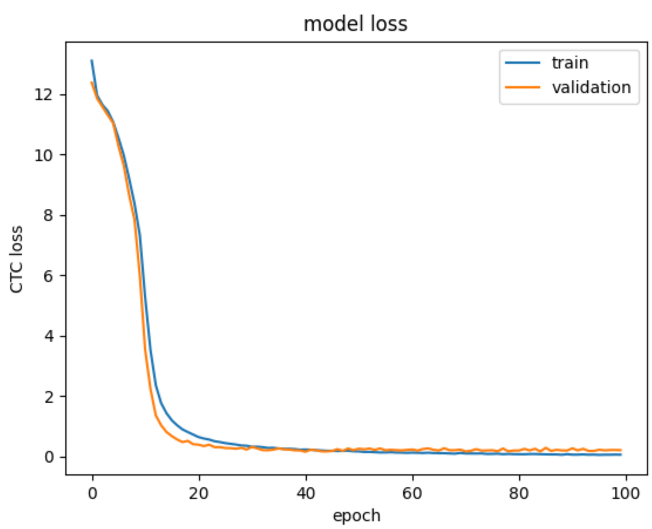
- **Build and train the model**
    - Train the model using **tensorflow.keras**.
    - The prediction result(labels) is the output of ‘dense2’ layer.

```python
# Build the model --------------------------------------------------------------------------------
model = util.build_model(characters=characters, img_width=img_width, img_height=img_height, max_length=max_length)
plot_model(model, show_shapes=True, show_layer_names=True,to_file='model.png')
model.summary()
save_dir=f"./Saved_Model/"
model_filepath = "model1.h5"

es = keras.callbacks.EarlyStopping(monitor='val_loss', patience=100, restore_best_weights=True)  # Add early stopping
checkpoint = ModelCheckpoint(os.path.join(save_dir, model_filepath), monitor='val_loss', verbose=0, save_best_only=True, mode='min')

# Train the model --------------------------------------------------------------------------------
history = model.fit(train_data_generator,
                    validation_data=valid_data_generator,
                    epochs=train_epoch,
                    callbacks=[es, checkpoint])

model = load_model(f"{save_dir}/{model_filepath}", custom_objects={'CTCLayer' : util.CTCLayer})
prediction_model = keras.models.Model(model.get_layer(name='input_data').input, model.get_layer(name='dense2').output)
prediction_model.summary()
```



5. **My experiment result**

	batch_size	img size	Kaggle performance
Test 1	128	(72x100)	0.8152
Test 2	64	(72x100)	0.9514
Test 3	16	(72x100)	**0.9620**
Test 4	8	(72x100)	**0.9634**
Test 5	4	(72x100)	0.9570
Test 6	2	(72x100)	0.9548
Test 7	8	(100x100)	0.9534

6. **Compare with different method**

- **CNN + FC Classifier**

  1. Crop the image around the characters and flip it horizontally to align the time dimension with the image width.
  2. Apply a **Convolutional Neural Network (CNN)** to extract features from the cropped and flipped image.
  3. Reshape the extracted features to split them into 5 time-steps, corresponding to different segments of the image.
  4. Utilize a **Fully Connected (FC) Classifier** to predict 5 characters at each time-step.
  5. Obtain the probability distribution of each character's presence at each time-step from the FC Classifier's output.

  - **Model Architecture**

```
Model: "ocr_classifier_based_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
image (InputLayer)           [(None, 200, 50, 1)]      0
_________________________________________________________________
Conv1 (Conv2D)               (None, 200, 50, 32)       320
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 100, 25, 32)       0
_________________________________________________________________
Conv2 (Conv2D)               (None, 100, 25, 64)       18496
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 50, 12, 64)        0
_________________________________________________________________
reshape (Reshape)            (None, 5, 7680)           0
_________________________________________________________________
dense1 (Dense)               (None, 5, 256)            1966336
_________________________________________________________________
dense2 (Dense)               (None, 5, 64)             16448
_________________________________________________________________
dense3 (Dense)               (None, 5, 19)             1235
=================================================================
Total params: 2,002,835
Trainable params: 2,002,835
Non-trainable params: 0
_________________________________________________________________
```

- The performance is not as good as CNN+RNN+CTCloss model in the previous part.

7. Reference

Keras documentation: OCR model for reading Captchas
Keras documentation
https://keras.io/examples/vision/captcha_ocr/