Министерство науки и высшего образования

Российский Федерации

Федеральное государственное бюджетное образовательное

учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

**АУГМЕНТАЦИЯ ИЗОБРАЖЕНИЙ**

Практическая работа №3

по дисциплине «Основы проектной деятельности»

Обучающаяся гр. 430-2:

\_\_\_\_\_\_\_\_\_\_\_\_\_ Лузинсан А.А.

\_\_.\_\_.2021

Руководитель

Преподаватель каф. БИС:

\_\_\_\_\_\_\_\_\_\_\_ Светлаков М.О.

\_\_.\_\_.2021

Томск 2021

**Оглавление**

[**1 Цель работы** 3](#_heading=h.gjdgxs)

[**2 Выполнение задания** 4](#_heading=h.30j0zll)

[**3 Листинг программы** 6](#_heading=h.1fob9te)

[**4 Результат** 10](#_heading=h.2et92p0)

[**5 Вывод** 13](#_heading=h.tyjcwt)

[**6 Источники** 14](#_heading=h.1t3h5sf)

1. **1 Цель работы**

Повысить число классов сверточной нейронной сети из предыдущей практики до 120-150. Добавить аугментацию изображений. Попробовать добавить регуляризацию.

1. **2 Выполнение задания**

По заданию, требовалось добавить аугментацию. В качестве аугментации используется resize\_and\_rescale. Также прописан data\_augmentation, который поворачивает и искажает картинку, но в тестовой выборке была применена только первый способ аугментации. Пример одного обработанного изображения представлен на рисунке 2.2. Исходная картинка представлена на рисунке 2.1.

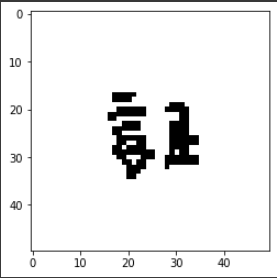


Рисунок 2.1 – Рисунок до аугментации

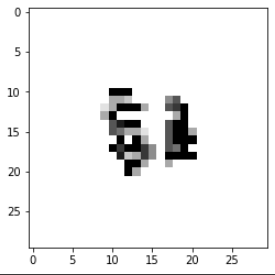


Рисунок 2.2 – Рисунок после аугментации

Структура сверточной сети изображена на рисунке 2.3.

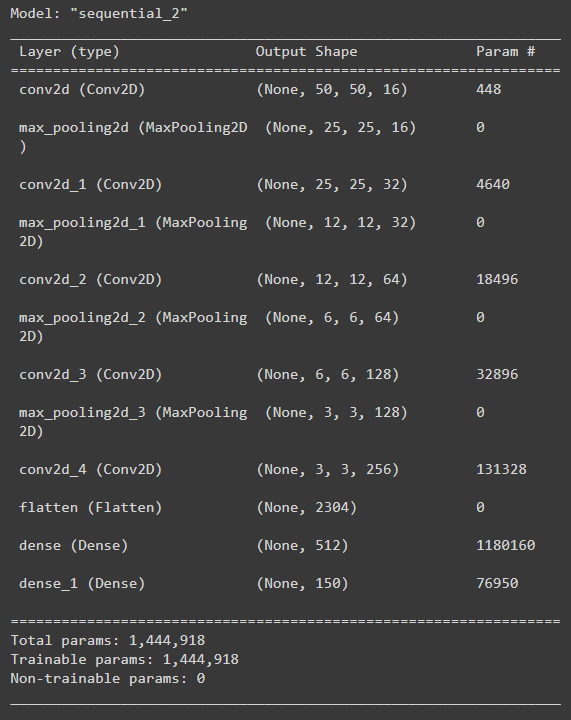


Рисунок 2.3 – Структура сверточной нейронной сети

Также была добавлена регуляризация весов нейронной сети в слой Dense(512, activation=’elu’, kernel\_regularizer=regularizers.l2(0.001) ), что означает, что каждый коэффициент в весовой матрице слоя будет добавлять (0,001\*weight\_coefficient\_value2) к лоссу сети.

1. **3 Листинг программы**

**import** numpy **as** np

**import** time

**import** PIL.Image **as** Image

**import** matplotlib.pylab **as** plt

**import** tensorflow **as** tf

**import** os

**import** zipfile

**import** shutil

**!**git clone https://github.com/AI-FREE-Team/Traditional-Chinese-Handwriting-Dataset.git

OutputFolder **=** '/content/Handwritten\_Data'

**!**rm -rf '/content/Handwritten\_Data'

SIZE **=** 150 *# рассматриваем 150 классов*

CompressedFiles **=** []

os**.**chdir('/content/Traditional-Chinese-Handwriting-Dataset/data')

**for** item **in** os**.**listdir():

**if** item**.**endswith('.zip'): *# Check for ".zip" extension.*

file\_path **=** os**.**path**.**abspath(item) *# Get full path of the compressed file.*

CompressedFiles**.**append(file\_path)

**for** file **in** CompressedFiles:

*# Construct a ZipFile object with the filename, and then extract it.*

zip\_ref **=** zipfile**.**ZipFile(file)**.**extractall(OutputFolder)

source\_path **=** OutputFolder **+** '/cleaned\_data(50\_50)'

img\_list **=** os**.**listdir(source\_path)

**for** img **in** img\_list:

shutil**.**move(source\_path **+** '/' **+** img, OutputFolder) *# Move a file to another location.*

shutil**.**rmtree(OutputFolder **+** '/cleaned\_data(50\_50)')

ImageList **=** os**.**listdir(OutputFolder)

ImageList **=** [img **for** img **in** ImageList **if** len(img)**>**1]

WordList **=** list(set([w**.**split('\_')[0] **for** w **in** ImageList]))[:SIZE]

**for** w **in** WordList:

**try**:

os**.**chdir(OutputFolder) *# Change the current working directory to OutputPath.*

os**.**mkdir(w) *# Create the new word folder in OutputPath.*

MoveList **=** [img **for** img **in** ImageList **if** w **in** img]

**except**:

os**.**chdir(OutputFolder)

MoveList **=** [img **for** img **in** ImageList **if** w **in** img ]

**finally**:

**for** img **in** MoveList:

old\_path **=** OutputFolder **+** '/' **+** img

new\_path **=** OutputFolder **+** '/' **+** w **+** '/' **+** img

shutil**.**move( old\_path, new\_path )

print( 'Data Deployment completed.' )

a**=**0

b**=**0

**for** item **in** os**.**listdir(OutputFolder):

**if** (os**.**path**.**isdir(item)):

a **+=** 1

**for** i **in** os**.**listdir(OutputFolder **+** '/' **+** item):

b **+=**1

image\_generator **=** tf**.**keras**.**preprocessing**.**image**.**ImageDataGenerator(validation\_split**=**0.1)

train\_dataset **=** image\_generator**.**flow\_from\_directory(str(OutputFolder), class\_mode**=**'sparse', batch\_size**=**10, target\_size**=**(50, 50), subset**=**'training')

valid\_dataset **=** image\_generator**.**flow\_from\_directory(str(OutputFolder), class\_mode**=**'sparse', batch\_size**=**10, target\_size**=**(50, 50), subset**=**'validation')

data\_augmentation **=** tf**.**keras**.**Sequential([ tf**.**keras**.**layers**.**RandomFlip("horizontal\_and\_vertical"), tf**.**keras**.**layers**.**RandomRotation(0.2), ])

IMG\_SIZE **=** 30

resize\_and\_rescale **=** tf**.**keras**.**Sequential([ tf**.**keras**.**layers**.**Resizing(IMG\_SIZE, IMG\_SIZE),

tf**.**keras**.**layers**.**Rescaling(1.**/**255) ])

**for** image\_b, label\_b **in** train\_dataset:

plt**.**imshow(image\_b[1])

image\_batch **=** resize\_and\_rescale(image\_b)

label\_batch **=** label\_b

**break**

plt**.**imshow(image\_batch[1])

model **=** tf**.**keras**.**Sequential([

tf**.**keras**.**layers**.**Conv2D(filters**=**16, kernel\_size**=**3, activation**=**'relu', padding**=** 'same' , input\_shape**=**(50,50,3)),

tf**.**keras**.**layers**.**MaxPooling2D(pool\_size**=**(2,2), strides**=**2),

tf**.**keras**.**layers**.**Conv2D(filters**=**32, kernel\_size**=**3, activation**=**'relu', padding**=** 'same'),

tf**.**keras**.**layers**.**MaxPooling2D(pool\_size**=**(2,2), strides**=**2),

tf**.**keras**.**layers**.**Conv2D(filters**=**64, kernel\_size**=**3, activation**=**'relu', padding**=** 'same'),

tf**.**keras**.**layers**.**MaxPooling2D(pool\_size**=**(2,2), strides**=**2),

tf**.**keras**.**layers**.**Conv2D(filters**=**128, kernel\_size**=**2, activation**=**'relu', padding**=** 'same'),

tf**.**keras**.**layers**.**MaxPooling2D(pool\_size**=**(2,2), strides**=**2),

tf**.**keras**.**layers**.**Conv2D(filters**=**256, kernel\_size**=**2, activation**=**'relu', padding**=** 'same'),

tf**.**keras**.**layers**.**Flatten(),

tf**.**keras**.**layers**.**Dense(512, activation**=**'relu', kernel\_regularizer**=**tf**.**keras**.**regularizers**.**l2(0.001)),

tf**.**keras**.**layers**.**Dense(SIZE, activation**=**'softmax')

])

model**.**summary()

lr\_schedule **=** tf**.**keras**.**optimizers**.**schedules**.**InverseTimeDecay(1.8, decay\_steps **=** 705**\***5, decay\_rate **=** 0.1, staircase **=** **False**)

model**.**compile( optimizer**=**tf**.**keras**.**optimizers**.**Adadelta(lr\_schedule),

loss**=**tf**.**keras**.**losses**.**SparseCategoricalCrossentropy(),

metrics**=**['accuracy']

)

EPOCHS **=** 20

history **=** model**.**fit(train\_dataset,

validation\_data**=**valid\_dataset,

epochs**=**EPOCHS)

acc **=** history**.**history['accuracy']

val\_acc **=** history**.**history['val\_accuracy']

loss **=** history**.**history['loss']

val\_loss **=** history**.**history['val\_loss']

plt**.**figure(figsize**=**(10, 10))

plt**.**subplot(2, 1, 1)

plt**.**plot(acc, label**=**'Training Accuracy')

plt**.**plot(val\_acc, label**=**'Validation Accuracy')

plt**.**legend(loc**=**'lower right')

plt**.**ylabel('Accuracy')

plt**.**ylim([min(plt**.**ylim()),1])

plt**.**title('Training and Validation Accuracy')

plt**.**subplot(2, 1, 2)

plt**.**plot(loss, label**=**'Training Loss')

plt**.**plot(val\_loss, label**=**'Validation Loss')

plt**.**legend(loc**=**'upper right')

plt**.**ylabel('Cross Entropy')

plt**.**ylim([min(plt**.**ylim()),max(plt**.**ylim())])

plt**.**title('Training and Validation Loss')

plt**.**xlabel('epoch')

plt**.**show()

lr\_schedule **=** tf**.**keras**.**optimizers**.**schedules**.**InverseTimeDecay(0.5, decay\_steps **=** 705**\***5, decay\_rate **=** 0.1, staircase **=** **False**)

model**.**compile(

optimizer**=**tf**.**keras**.**optimizers**.**Adadelta(lr\_schedule),

loss**=**tf**.**keras**.**losses**.**SparseCategoricalCrossentropy(),

metrics**=**['accuracy']

)

EPOCHS **=** 20

history **=** model**.**fit(train\_dataset,

validation\_data**=**valid\_dataset,

epochs**=**EPOCHS)

acc **=** history**.**history['accuracy']

val\_acc **=** history**.**history['val\_accuracy']

loss **=** history**.**history['loss']

val\_loss **=** history**.**history['val\_loss']

plt**.**figure(figsize**=**(10, 10))

plt**.**subplot(2, 1, 1)

plt**.**plot(acc, label**=**'Training Accuracy')

plt**.**plot(val\_acc, label**=**'Validation Accuracy')

plt**.**legend(loc**=**'lower right')

plt**.**ylabel('Accuracy')

plt**.**ylim([min(plt**.**ylim()),1])

plt**.**title('Training and Validation Accuracy')

plt**.**subplot(2, 1, 2)

plt**.**plot(loss, label**=**'Training Loss')

plt**.**plot(val\_loss, label**=**'Validation Loss')

plt**.**legend(loc**=**'upper right')

plt**.**ylabel('Cross Entropy')

plt**.**ylim([min(plt**.**ylim()),max(plt**.**ylim())])

plt**.**title('Training and Validation Loss')

plt**.**xlabel('epoch')

plt**.**show()

1. **4 Результат**

Во время обучения применялось снижение скорости обучения, однако должных результатов добиться не удалось, и нейронная сеть обучалась в два этапа. Первый этап представлен на рисунке 4.1. Второй же этап – на рисунке 4.2. Оба процесса в табличной форме представлены на рисунках 4.3-4.4. В ходе обучения на некоторой эпохе удалось достичь минимальных потерь только со значением val\_loss=0.32, однако запечатлеть это событие не удалось.

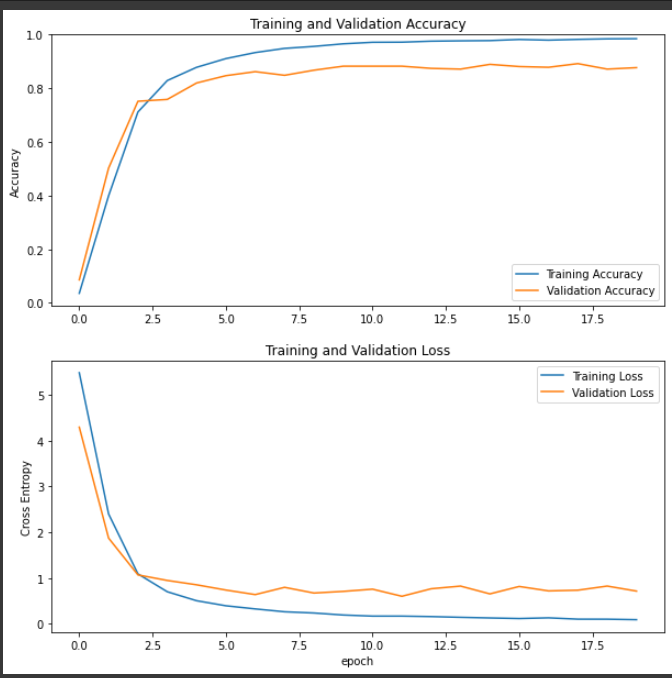


Рисунок 4.1 – График данных тренировки и проверки обучающейся нейронной сети на первой стадии обучения

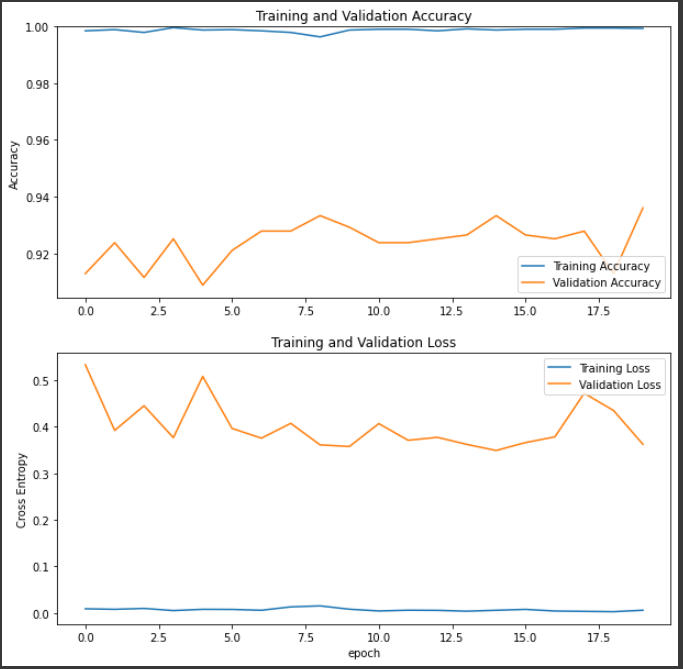


Рисунок 4.2 – График данных тренировки и проверки обучающейся нейронной сети на второй стадии обучения

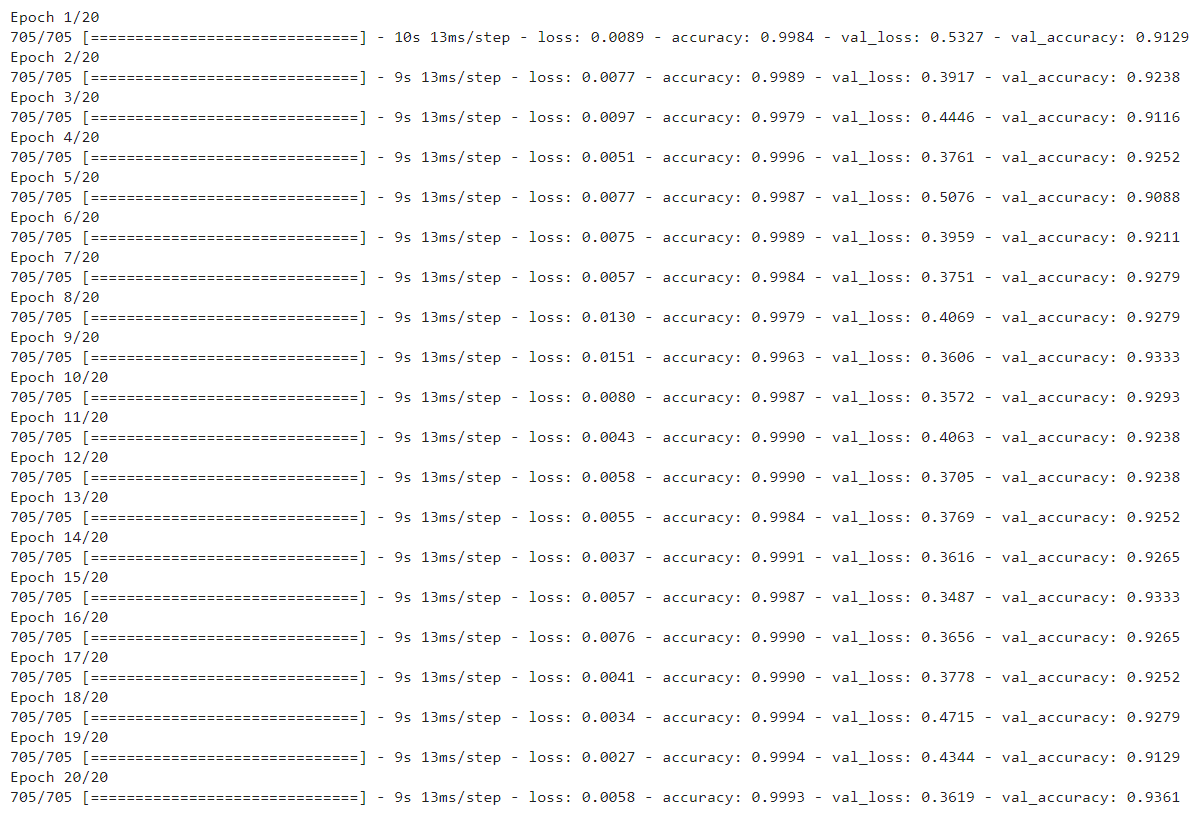


Рисунок 4.3 – Таблица данных первого этапа обучения сверточной нейронной сети

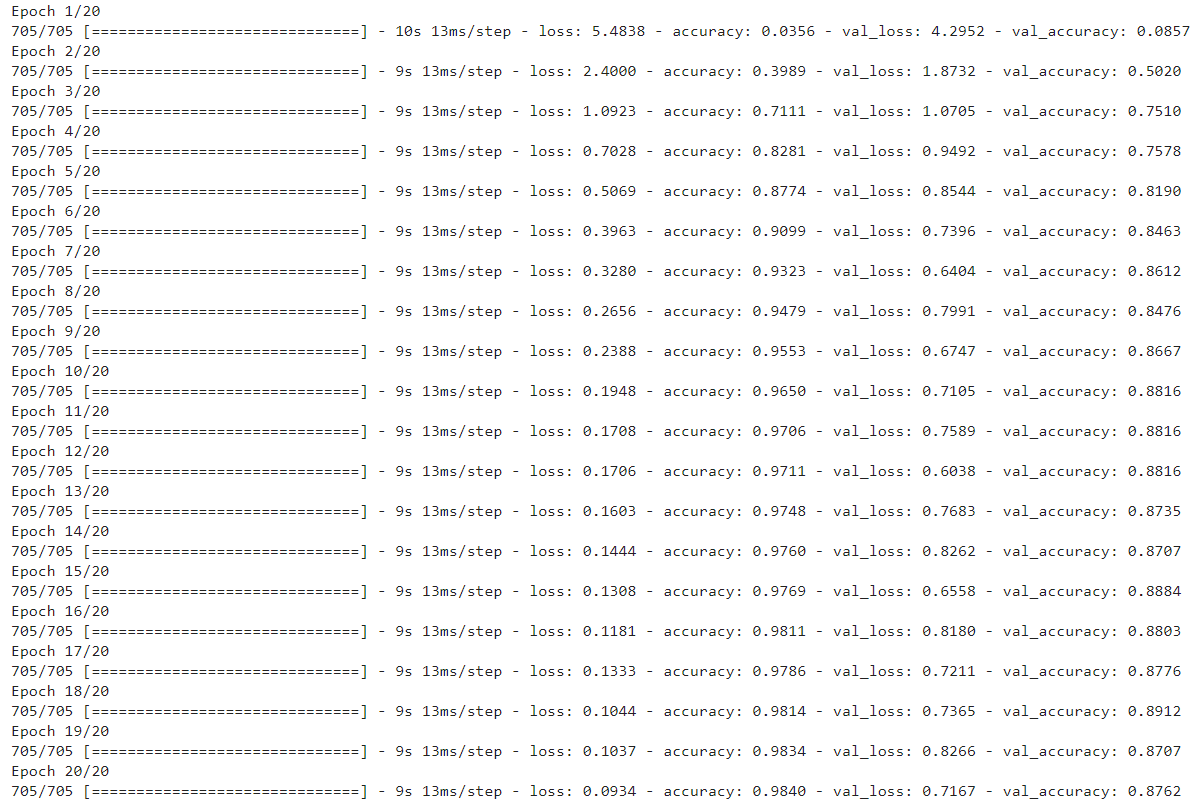


Рисунок 4.4 – Таблица данных второго этапа обучения сверточной нейронной сети

1. **5 Вывод**

Я повысила число классов сверточной нейронной сети из предыдущей практики до 150 классов, а также добавила аугментацию изображений и регуляризацию весов нейронной сети.

1. **6 Источники**
2. Исходный ноутбук: <https://colab.research.google.com/drive/1vMV1HCKJiHNQGT6-riiBnWehNCI4X8gz#scrollTo=WucSLXxZdpuc>
3. Вспомогательный ноутбук с аугментациями данных: https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/data\_augmentation.ipynb#scrollTo=r1Bt7w5VhVDY
4. GitHub выполненного проекта: <https://github.com/Luzinsan/augmentation_and_regularizer/blob/main/Copy_of_Handwritten_Chinese_v3.ipynb>
5. TensorFlow Layers: <https://www.tensorflow.org/api_docs/python/tf/keras/layers?hl=ru>
6. TensorFlow Activations: <https://www.tensorflow.org/api_docs/python/tf/keras/activations?hl=ru>
7. TensorFlow Optimizers: <https://www.tensorflow.org/api_docs/python/tf/optimizers?hl=ru>
8. TensorFlow Losses: <https://www.tensorflow.org/api_docs/python/tf/losses?hl=ru>