УДК: 004.85

Сейтказиева Н.С.

окутуучу

И. Арабаев атындагы Кыргыз мамлекеттик университети

Бишкек ш.

s.nazgul.s@mail.ru

Иса кызы Б.

магистрант

И. Арабаев атындагы Кыргыз мамлекеттик университети

Бишкек ш.

Джумачалова Ч.Т.

магистрант

И. Арабаев атындагы Кыргыз мамлекеттик университети

Бишкек ш.

Сабитов Б.Р.

физика-математика илимдеринин кандидаты, доцент

Ж. Баласагын атындагы Кыргыз Улуттук университети

Бишкек ш.

"КОЛДОНМО ТАПШЫРМАЛАРДЫ БОЛЖОЛДОО ҮЧҮН ТЕРЕҢ ОКУТУУДА МААЛЫМАТТАРДЫ КӨБӨЙТҮҮ ЫКМАСЫН КОЛДОНУУ"

Сейтказиева Н.С.

преподаватель

Кыргызский государственный университет имени И. Арабаева

г. Бишкек

s.nazgul.s@mail.ru

Иса кызы Б.

магистрант

Кыргызский государственный университет имени И. Арабаева

г. Бишкек

Джумачалова Ч.Т.

магистрант

Кыргызский государственный университет имени И. Арабаева

г. Бишкек

Сабитов Б.Р.

кандидат физико-математических наук, доцент

Кыргызский Национальный Университет имени Ж. Баласагына

г.Бишкек

«ИСПОЛЬЗОВАНИЕ МЕТОДА УВЕЛИЧЕНИЯ ДАННЫХ В ГЛУБОКОМ ОБУЧЕНИИ ДЛЯ ЗАДАЧ ПРОГНОЗИРОВАНИЯ ПРИКЛАДНЫХ ЗАДАЧ»

Seitkazieva N.S.

teacher

Kyrgyz State University named after I. Arabaev

Bishkek c.

s.nazgul.s@mail.ru

Isa kizi B.

Master

Kyrgyz State University named after I. Arabaev

Bishkek c.

Jumachalova Ch.T.

Master

Kyrgyz State University named after I. Arabaev

Bishkek c.

Sabitov B.R.

Ph.D., Associate Professor

Kyrgyz National University named after J. Balasagyn

Bishkek c.

"USING THE DATA AUGMENTATION METHOD IN DEEP LEARNING FOR PREDICTIVE TASKS OF APPLIED TASKS"

Аннотация.

Ачык системалардан алынган маалыматтардын аз топтому менен RGB сүрөтүн таануу маселеси изилденген. Моделдер нейрон тармактарынын архитектурасынын негизинде курулган. Кайра даярдалган моделдер үчүн Dropout технологиясы колдонулган. Моделдерди куруу процесси Python китепканаларына негизделет. Ошондой эле биз TensorFlow китепканасын жана сүрөттөр менен иштөө үчүн башка Python китепканаларын импорттойбуз.

Төмөндө биз Анаконда-Юпитерде бул тапшырманы ишке ашыруу процессин көрсөтөбүз. Программалардын мүчүлүштүктөрүн оңдоонун бүт процесси макаланын материалдарын көрсөтүүнүн ыңгайлуулугу үчүн сүрөттөлгөн. Python китепканасынын программаларынын агымы колдонулат. Берилиштер топтомун TensorFlow китепканаларынан же ачык булак системаларынан жүктөп алса болот.

Аннотация.

Исследуется задача распознавания изображений в формате RGB с небольшим набором данных из открытых систем. Строится модели, основанные на архитектурах нейронных сетей. Для переобученных моделей использовано технология Dropout. Весь процесс построения моделей будет опираться на библиотеки Python.Импортируем, также библиотеку TensorFlow и других библиотек Python для работы с изображениями. Ниже, мы демонстрируем процесс реализации данной задачи на Anaconda-Jupyter. Описан весь процесс отладки программ для удобства изложения материалов статьи. Используется поток программ-библиотек Python. Набор данных можно скачать из библиотек TensorFlow или из открытых систем.

Annotation.

The problem of recognizing images in RGB format with a small set of data from open systems is being investigated. Models based on neural network architectures are being built. Dropout technology is used for retrained models. The whole process of building models will rely on Python libraries. We also import the TensorFlow library and other Python libraries for working with images. Below, we demonstrate the process of implementing this task on Anaconda-Jupiter. The entire process of debugging programs is described for the convenience of presenting the materials of the article. A stream of Python library programs is used. The dataset can be downloaded from TensorFlow libraries or from open systems.

Негизги сөздөр: терең үйрөнүү, нейрон тармактары, моделди иштеп чыгуу, машинаны үйрөнүү, алгоритм, технология, маалымат базалары, таануу ыкмасы, конволюциялык нейрон тармактары.

Ключевые слова: глубокое обучение, нейронные сети, разработке модели, машинное обучение, алгоритм, технология, базы данных, метод распознавания, сверточные нейронные сети.

Keywords: deep learning, neural networks, model development, machine learning, algorithm, technology, databases, recognition method, convolutional neural networks.

В данной работе исследуется основные операции машинного обучения, которая создает весь процесс построения и обучения модели для прогнозирования прикладных задач. Ниже

описано ключевые моменты для создания модели в машинном обучении. Она состоит из следующих:

• Загрузка, изучение и обработка данных;

• Создать входной набор данных для нейронной сети;

• Построение архитектуры нейронной сети;

• Обучение модели;

• Тестирование модели;

• Улучшение модели.

Формируется также каталог данных для обучения модели. Данный каталог содержит множество изображений различных цветов, в цветовых каналах. Многие изображения имеют четкое представление в трехцветном изображении. Всего изображений 3700 из 5 типов изображений.

Дальнейшие действия будут направлены из какого источника данных скачивать данные, мы остановились TensorFlow. Для скачивания и подготовка набора данных к реализации пишем следующий код в Python. Приведем процесс вызова директорий.

```
[ ] import pathlib
    dataset_url = "https://storage.googleapis.com/
    download.tensorflow.org/
    example_images/flower_photos.tgz"
    data_dir =
    tf.keras.utils.get_file('flower_photos', origin=dataset_url, untar=True)
    data_dir =
    pathlib.Path(data_dir)
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/
228818944/228813984 [==============================] - 2s 0us/step
```

В данном случае набор данных tf_flowers размещен на специальном каталоге TensorFlow. Он будет автоматически загружен на наш локальный каталог данных, если мы укажем каталог и адрес расположения набора данных.

```
/home/kbuilder/tensorflow_datasets/
tf_flowers/3.0.1. Subsequent calls will reuse this data.
```

Набор данных цветов состоит из пяти классов.

```
[2] import pathlib
    dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
    data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url, untar=True)
    data_dir = pathlib.Path(data_dir)
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz
228818944/228813984 [==============================] - 1s 0us/step
```

Будем изучать цветные изображения случайно выбранное изображение выглядит следующим образом. Предварительно мы выбрали среду где программа будет реализовывается. Данное программа была реализовано в Colaboratory. При разработке модели рекомендуется использовать разделение валидации. Давайте использовать 80% изображений для обучения и 20% для проверки.

```
[9] train_ds = tf.keras.preprocessing.image_dataset_from_directory(
      data_dir,
      validation_split=0.2,
      subset="training",
      seed=123,
      image_size=(img_height, img_width),
      batch_size=batch_size)
```

```
Found 3670 files belonging to 5 classes.
Using 2936 files for training.
```

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

При реализации мы обнаруживаем, что всего найдено 3670 файлов-изображений, принадлежащих 5 разным классам. Для тестирования модели, необходимо выделит отдельный набор данных. В данном случае мы будем использовать 734 файлов для проверки модели. Теперь давайте обучим нашу модель, используя эти наборы данных, мгновенно передав их в model.fit. Если хотите, вы также можете вручную перебирать набор данных и получать пакеты изображений:

```
[ ] for image_batch, labels_batch in train_ds:
      print(image_batch.shape)
      print(labels_batch.shape)
      break
```

Image_batch - это тензор формы (32, 180, 180, 3). Это пакет из 32 изображений формы 180x180x3 (последнее измерение относится к цветовым каналам RGB). Label_batch - это тензор формы (32,), это соответствующие метки для 32 изображений.

Вы можете вызвать .numpy () для тензоров image_batch и labels_batch, чтобы преобразовать их в numpy.ndarray.

Настройте набор данных для повышения производительности

Обязательно используйте буферизованную предварительную выборку, чтобы вы могли передавать данные с диска без блокировки ввода-вывода. Это два важных метода, которые вы должны использовать при загрузке данных [7].

Dataset.cache () сохраняет изображения в памяти после их загрузки с диска в течение первой эпохи. Это гарантирует, что набор данных не станет узким местом при обучении вашей модели. Если ваш набор данных слишком велик для размещения в памяти, вы также можете использовать этот метод для создания производительного кеша на диске.

Dataset.prefetch () перекрывает предварительную обработку данных и выполнение модели во время обучения.

Таким образом мы описали два метода которые сохраняют готовые изображения, а также о том, как кэшировать данные на диск, в руководстве по производительности данных.

```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

Стандартизация данных

Значения канала RGB находятся в диапазоне [0, 255]. Это не идеально для нейронной сети; в общем, вы должны стремиться к тому, чтобы ваши входные значения были небольшими. Здесь вы стандартизируете значения, чтобы они находились в диапазоне [0, 1], используя слой изменения масштаба.

```
[ ] normalization_layer = layers.experimental.preprocessing.Rescaling(1./255)
```

Многие программы и слои предварительной обработки Keras, представленные в этом разделе, в настоящее время являются экспериментальными и могут измениться. Есть два способа использовать этот слой. Давайте сначала мы будем использовать его к набору данных, вызвав map:

```
[26] normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
     image_batch, labels_batch = next(iter(normalized_ds))
     first_image = image_batch[0]
     # Notice the pixels values are now in `[0,1]`.
     print(np.min(first_image), np.max(first_image))

     0.0 1.0
```

Или вы можете включить слой в определение модели, что упростит развертывание. Воспользуемся здесь вторым подходом.

Вообще меня размер изображений, используя аргумент image_size в image_dataset_from_directory можем создать новое изображение. Если вы хотите включить логику изменения размера в свою модель, вы можете использовать слой Resizing.

Создание модели

Ниже создан модель, которая состоит из трех блоков свертки с максимальным уровнем пула в каждом из них. Есть полностью связанный слой со 128 модулями поверх него, который активируется функцией активации relu. Эта модель не была настроена для обеспечения высокой точности, цель этой работы - показать стандартный подход.

```
num_classes = 5

model = Sequential([
  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

Вот компиляция модели, вид нашей модели выглядит таким образом.

В данном случае мы выбираем мощный оптимизатор Адам. Для корректной оценки модели в данном случае выберем меру оценку по специальному методу кросс энтропии. Для оценки точности модели, чтобы просмотреть точность обучения и проверки для каждой эпохи обучения, передадим аргумент метрики [8].

```
[28] model.compile(optimizer='adam',
                   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                   metrics=['accuracy'])
```

Краткое описание модели

Теперь приступим к изучению всех слоев. Сначала изучим все слои все слои за слоем в полной нейронной сети, используя метод, который описано ниже в коде:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_1 (Rescaling)      (None, 180, 180, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 180, 180, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 90, 90, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 90, 90, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 45, 45, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 45, 45, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 30976)             0
_________________________________________________________________
dense (Dense)                (None, 128)               3965056
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 645
=================================================================
Total params: 3,989,285
Trainable params: 3,989,285
Non-trainable params: 0
_________________________________________________________________
```

Обучение нашей модели происходить следующим образом. Построим наш модель со следующими параметрами нейронной сети. Вот полный код, который, отвечает за процесс обучения модели и процесс оценки ошибки модели.

```
epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/10
92/92 [==============================] - 36s 41ms/step - loss: 1.5949 - accuracy: 0.3100 - val_loss: 1.0931 - val_accuracy: 0.5531
Epoch 2/10
92/92 [==============================] - 2s 20ms/step - loss: 1.0329 - accuracy: 0.5932 - val_loss: 1.0057 - val_accuracy: 0.6035
Epoch 3/10
92/92 [==============================] - 2s 20ms/step - loss: 0.8822 - accuracy: 0.6634 - val_loss: 0.9895 - val_accuracy: 0.6185
Epoch 4/10
92/92 [==============================] - 2s 20ms/step - loss: 0.6587 - accuracy: 0.7453 - val_loss: 1.0072 - val_accuracy: 0.6213
Epoch 5/10
92/92 [==============================] - 2s 20ms/step - loss: 0.4947 - accuracy: 0.8293 - val_loss: 1.0825 - val_accuracy: 0.6253
Epoch 6/10
92/92 [==============================] - 2s 20ms/step - loss: 0.3011 - accuracy: 0.9056 - val_loss: 1.2336 - val_accuracy: 0.6390
Epoch 7/10
92/92 [==============================] - 2s 20ms/step - loss: 0.1381 - accuracy: 0.9629 - val_loss: 1.3245 - val_accuracy: 0.6253
Epoch 8/10
92/92 [==============================] - 2s 20ms/step - loss: 0.0709 - accuracy: 0.9820 - val_loss: 1.6163 - val_accuracy: 0.6471
Epoch 9/10
92/92 [==============================] - 2s 20ms/step - loss: 0.0335 - accuracy: 0.9910 - val_loss: 2.0855 - val_accuracy: 0.6253
Epoch 10/10
92/92 [==============================] - 2s 20ms/step - loss: 0.0226 - accuracy: 0.9942 - val_loss: 1.8312 - val_accuracy: 0.6035
```

Теперь давайте вернемся к процессу визуализации. Покажем процесс визуализации, как результат обучения. Создадим графики потерь и точности на наборах для обучения и проверки.

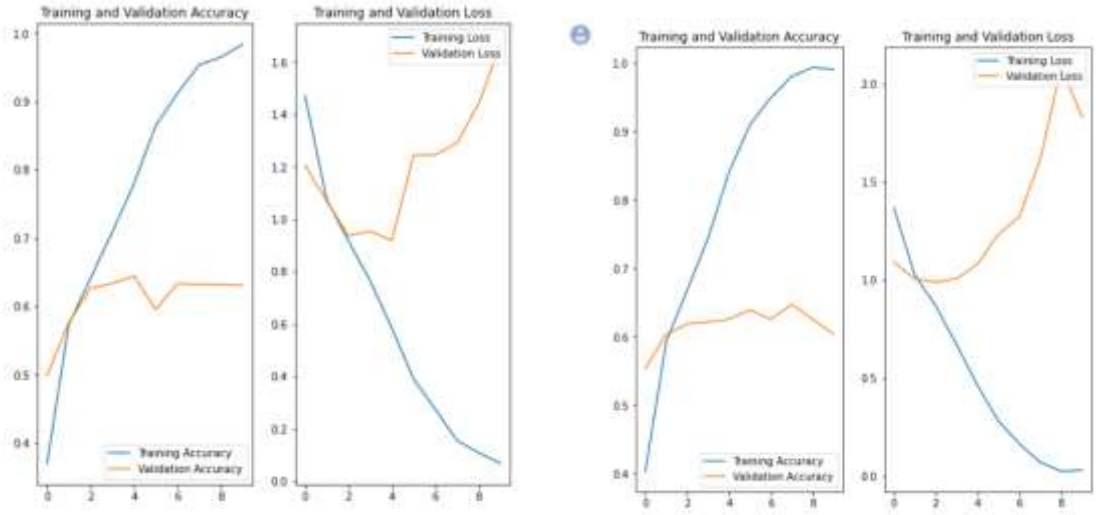

Рис. 1. Результаты обученных моделей на данных обучения и тестирования.

Из этого графика мы видим, что точность обучения и точность проверки значительно ниже, и модель достигла только около 60% точности на проверочном тестовом наборе наборе. Это не лучший вариант построения модели. Давайте посмотрим, почему так получилось не так, и попробуем повысить общую производительность модели.

Переобучение модели

На графиках выше точность обучения линейно увеличивается со временем, тогда как точность проверки в процессе обучения составляет около 60%. В данном случае мы видим, что разница в точности между точностью обучения и проверки - признак переобучения. Когда имеется небольшое количество обучающих примеров, модель иногда учится на шуме или нежелательных деталях обучающих примеров - до такой степени, что это отрицательно влияет на производительность модели на новых примерах. Это явление называется переобучением или переоснащение. Данный момент объясняется следующим образом, модели будет сложно обобщить на новом наборе данных. Есть несколько способов борьбы с переобучением в

тренировочном процессе. Можно выйти из положения если мы будем использовать увеличение данных и добавим отсев Dropout в нашу обучаемому модели.

Увеличение данных

Технология переобучение обычно происходит при небольшом количестве обучающих данных т.е. на малых данных модель не успевает обучаться. Доя увеличении данных, мы будем опираться на случай, используется подход к генерации дополнительных обучающих данных из ваших существующих примеров путем их дополнения с помощью случайных преобразований, которые дают правдоподобно выглядящие изображения. Это помогает раскрыть модель большему количеству аспектов данных и лучше обобщить.

Ее мы будем реализовывать следующим образом, увеличение данных, используя слои из tf.keras.layers.experimental.preprocessing. Теперь они могут быть включены в нашу модель, как и другие слои, а потом запускается на графическом процессоре GPU, что очень важно при работе с графическими файлами. Следующий код дает возможность — это делать с помощью специальной функции.

```
data_augmentation = keras.Sequential(
  [
    layers.experimental.preprocessing.RandomFlip("horizontal",
                                                 input_shape=(img_height,
                                                              img_width,
                                                              3)),
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1),
  ]
)
```

Обычно весь процесс визуализации данных при увеличения данных определяется следующим образом. Ниже даны несколько моментов дополненных изображений к исходным данным. Для этого ниже написано код преобразования и увеличить данные показав это на одном изображении. Мы описали способ увеличения данных на примере, и дали пример увеличения выглядит таким образом. Мы применяем увеличение данных к одному и тому же изображению несколько раз.

Метод регуляризации переобученных моделей.

Важным шагом для уменьшения переобучения моделей является ввести отсев в нейронной сети, это обычно делается с помощью функции Dropout. Для этой цели в процесс обучения вводится в сеть модели добавляют функцию Dropout, форму регуляризации.

В случае применения отсева или применения Dropout к слою, он случайным образом отбрасывает, некоторый выбранный нейрон, путем установки функции активации на ноль т.е. некоторое количество выходных данных из слоя во время процесса обучения превращается в ноль. Данный метод Dropout принимает дробное число в качестве входного значения в такой форме, как 0,1, 0,2, 0,4 и т. д. Это означает, что нейроны или случайное исключение 10%, 20% или 40% выходных нейронов из применяемого слоя в обучении не участвуют.

Вот созданное новая сеть, которую создадим как новую нейронную сеть с участием слоев Dropout, а затем обучим ее с помощью метода увеличения данных изображений.

Компиляция модели происходить обычным образом. Здесь определяется параметры модели и оценка точности модели, а также выбор оптимизаторов, которые отвечают за точность и устойчивость тестирования модели на тестовых данных. Вот компиляция модели. Можно теперь приступить и к ее обучению. Полное обучение модели с эпохой равным 15 является достаточным для обучения после увеличения данных. Всего выбрано в данном случае уже 15 эпох.

Визуализируем теперь результаты обучения

После увеличения данных, процесс обучения при применении этих дополнений данных и исключений переобучения должно быть намного меньше, чем раньше, а точность обучения и проверки должны быть более согласованными. Это один мощных подходов для регуляризации переобученных моделей.

Ниже представлен процесс обучения и тестирование модели после увеличения данных. Результаты получены в Jupyter и Colaboratory соответственно следующим образом. В графике показаны изменения и точность модели во время обучения и ошибка модели на проверочных и тестовых данных.

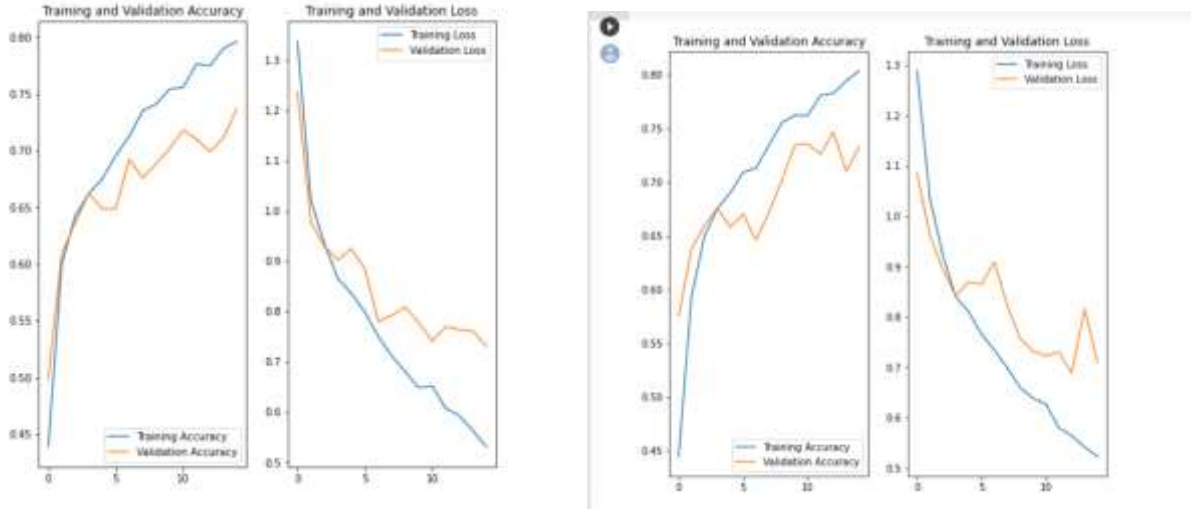

Рис.2. Результаты обучения модели после применения метода увеличения данных.

Мы полностью обучили модель и на тестовых данных получилось результат 99.2.%. Прогноз по новым данным, это хорошая проверка модели на устойчивость. Если модель хорошо себя зарекомендовал на обучающих данных, это не означает, что он также будет и на тестовых данных. Это обычно встречается, когда модель переобучена.

Заключение. Таким образом получили результаты тестирования нашей модели. Это изображение, скорее всего, принадлежит подсолнухам с 99.72 процентом уверенности. Таким образом мы угадали изображение почти 100 процентов. Это очень хороший результат. Результаты прогнозирования на основе нейронных технологий дали хороший результат прогноза. Чтобы, улучшит результаты можно использовать наиболее совершенные архитектуры нейронных сетей, например, методы распознавания с применением сверточных нейронных сетей.

ЛИТЕРАТУРА

1.Гудфеллоу Я., Бенджио И., Курвилль А. − Глубокое обучение, 2017 г.

2.Франсуа Шолле − Глубокое обучение на Python, 2018 г.

3.Орельен Жерон − Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow, 2018 г.

4.Ричард Саттон, Эндрю Барто − Обучение с подкреплением, 2017 г.

5.Ясер Абу-Мостафа, Малик Магдон-Исмаил, Сюань-Тянь Линь − Learning From Data, 2012 г.

6.Адриан Роузброк − Deep Learning for Computer Vision with Python, 2017 г.

7.Сабитов Б.Р., Сейтказиева Н.С., Кашкароева А.А. «БИНАРНАЯ ЗАДАЧА КЛАССИФИКАЦИИ БОЛЕЗНИ РАСТЕНИЙ С ПРИМЕНЕНИЕМ ТЕХНОЛОГИЙ ГЛУБОКОГО ОБУЧЕНИЯ» Наука, новые технологии и инновации Кыргызстана. 2022. № 7. С. 30-39.

8. Сабитов Б.Р., Сейтказиева Н.С., Кубанычбекова А.К. «ТЕХНОЛОГИИ ИНТЕГРАЦИИ МАШИННОГО ОБУЧЕНИЯ С ВЕБ ПРИЛОЖЕНИЯМИ», Современные проблемы механики. 2022. № 47 (1). С. 38-46.

Рецензент: Доктор PhD Ибраева Р.Т.