گزارش پروژه نهایی

اعضای گروه: فاطمه حورا حقیقت خواه - زهرا فارسی - میثم باوی

کنترلر ربات

کنترلر ربات، بر مبنای کنترلر اصلی ربات به نام mavic2pro_patrol توسعه داده شده است. در این کنترلر با استفاده یک PID controller، سعی می‌شود چهار مولفه اصلی حرکت ربات با توجه به ورودی کنترلر ثابت نگه داشته شوند. این چهار مولفه عبارت اند از: ارتفاع، میزان گردش حول محور x یا roll، میزان گردش حول محور y یا pitch و میزان گردش حول محور z یا yaw. برای حرکت به سمت جلو یا عقب، یک عدد مشخصی به عنوان pitch_disturbance با میزان pitch_acceleration جمع میشود؛ با توجه به منفی یا مثبت بودن این عدد، کنترلر PID ربات را به سمت جلو یا عقب خم می‌کند و همان‌طور نگه می‌دارد. در اثر این خم شدن و نیروی موتور ها، ربات به سمت جلو یا عقب حرکت می‌کند. برای تغییر جهت حرکت نیز طبق همین الگو، یک عدد مشخصی به نام yaw_disturbance، ورودی کنترلر PID را تغییر می‌دهد تا کنترلر به سرعت زاویه‌ای مورد نظر برسد. برای کنترلر ارتفاع هم یک ضریب کنترلر در مکعب اختلاف ارتفاع فعلی با ارتفاع هدف ضرب می‌شود و میزان نیروی تولید شده توسط همه موتورها را کنترلر می‌کند. تمام ضرایب کنترلر با آزمون و خطا به دست آمده اند.

برای انتقال ربات به نقطه هدف، پس از رسیدن به یک ارتفاع مشخص، با تعیین میزان yaw_disturbance و pitch_disturbance، ربات به سمت هدف می‌چرخد و حرکت می‌کند.

```python
if altitude > self.target_altitude - 1:
    # as soon as it reach the target altitude,
    # compute the disturbances to go to the given waypoints.
    if self.getTime() - t1 > 0.1:
        yaw_disturbance, pitch_disturbance = self.move_to_target(
            waypoints,
            verbose_target=True,
            # verbose_movement=True,
        )
        t1 = self.getTime()

roll_input = self.K_ROLL_P * clamp(roll, -1, value_max: 1) + roll_acceleration + roll_disturbance
pitch_input = self.K_PITCH_P * clamp(pitch, -1, value_max: 1) + pitch_acceleration + pitch_disturbance
yaw_input = yaw_disturbance
clamped_difference_altitude = clamp(self.target_altitude - altitude + self.K_VERTICAL_OFFSET, -1, value_max: 1)
vertical_input = self.K_VERTICAL_P * pow(clamped_difference_altitude, 3.0)

front_left_motor_input = self.K_VERTICAL_THRUST + vertical_input - yaw_input + pitch_input - roll_input
front_right_motor_input = self.K_VERTICAL_THRUST + vertical_input + yaw_input + pitch_input + roll_input
rear_left_motor_input = self.K_VERTICAL_THRUST + vertical_input + yaw_input - pitch_input - roll_input
rear_right_motor_input = self.K_VERTICAL_THRUST + vertical_input - yaw_input - pitch_input + roll_input

self.front_left_motor.setVelocity(front_left_motor_input)
self.front_right_motor.setVelocity(-front_right_motor_input)
self.rear_left_motor.setVelocity(-rear_left_motor_input)
self.rear_right_motor.setVelocity(rear_right_motor_input)
```

مقادیر yaw_disturbance و pitch_disturbance با توجه به فاصله تا هدف و زاویه با خط مستقیم تا هدف تعیین می‌شوند؛ yaw_disturbance ضریبی است از زاویه باقی مانده تا خط مستقیم تا هدف. pitch_disturbance هم با توجه به همین زاویه تعیین می‌شود. ایده‌آل این است که وقتی زاویه باقی مانده برزگ است، ربات به سمت جلو یا عقب خم نشود و سرجایش بماند؛ چون اگر حرکت کند، به جهت اشتباهی خواهد رفت. حتی خوب است کمی به عقب خم شود تا به پیچیدن ربات کمک کند. هنگامی هم که زاویه باقی مانده مقدار پایینی داشت، میخواهیم با تمام سرعت به جلو برویم و pitch_disturbance حداکثر مقدار را داشته باشد. به همین جهت، pitch_disturbance برابر است با لگاریتم قدرمطلق زاویه باقی‌مانده که در یک حداقل و حداکثر مشخصی clamp می‌شود؛ هر چه زاویه کمتر و به صفر نزدیک‌تر، مقدار pitch_disturbance نیز منفی‌تر (رو به جلو تر) و هر چه زاویه باقی مانده بیشتر، میزان pitch_disturbance مثبت و نزدیک‌تر به صفر (حداکثر برابر ۰.۱ تا ربات کمی به عقب خم شود).

```
# This will be in [-pi, pi]
self.target_position[2] = np.arctan2(
    self.target_position[1] - self.current_pose[1], self.target_position[0] - self.current_pose[0])
# This is now in [-2pi, 2pi]
angle_left = self.target_position[2] - self.current_pose[5]
# Normalize turn angle to [-pi, pi]
angle_left = (angle_left + np.pi) % (2 * np.pi) - np.pi

# Turn the robot to the left or to the right according the value and the sign of angle_left
yaw_disturbance = self.MAX_YAW_DISTURBANCE * angle_left / np.pi
# non-proportional and decreasing function
pitch_disturbance = clamp(
    np.log10(abs(angle_left)), self.MAX_PITCH_DISTURBANCE, value_max: 0.1)

if verbose_movement:
    distance_left = np.sqrt(((self.target_position[0] - self.current_pose[0]) ** 2) + (
        (self.target_position[1] - self.current_pose[1]) ** 2))
    print("remaning angle: {:.4f}, remaning distance: {:.4f}".format(
        *args: angle_left, distance_left))
return yaw_disturbance, pitch_disturbance
```

ربات به طور کلی چهار حالت کلی دارد؛ در حالت اولیه که SEARCHING نام دارد، ربات تک تک به سراغ نقاط هدف رفته و پس از رسیدن به یک هدف و گرفتن عکس از جعبه، تصمیم می‌گیرد کنار آن فرود بیاید یا به حرکت به نقطه بعدی ادامه دهد. این تصمیم با توجه به خروجی CNN برای عکس گرفته شده خواهد بود. هرگاه یک جعبه به عنوان هدف نهایی تشخیص داده شد و تصمیم به فرود گرفته شد، ربات با ورود به حالت GOING_TO_LAND_SITE، ابتدا مختصات نقطه‌ای 'کنار' جعبه را انتخاب می‌کند و آن را به عنوان نقطه هدف بعدی در نظر می‌گیرد و بدون کاهش ارتفاع به آن جا می‌رود. با ورود به این حالت، هر دو LED ربات نیز روشن می‌شوند.

پس از رسیدن به نقطه فرود، به حالت LANDING وارد می‌شود. در این حالت، ارتفاع هدف ربات به ارتفاع فرود (برابر با ۳ سانتی‌متر) تغییر می‌یابد. در این حالت ربات با ثابت نگه‌داشتن مختصات خود، به مرور نیروی موتورها را کم می‌کند تا به ارتفاع هدف برسد.

```
# if the robot is at the position with a precision of target_precision
if all([abs(x1 - x2) < self.target_precision for (x1, x2) in zip(self.target_position, self.current_pose[0:2])]):
    if self.state == GOING_TO_LAND_SITE:
        if verbose_target:
            print("Reached land site; Landing.")
        self.state = LANDING
        self.target_altitude = self.LAND_ALTITUDE

    elif self.state == SEARCHING and self.check_target_image():
        self.state = GOING_TO_LAND_SITE
        self.front_left_led.set(True)
        self.front_right_led.set(True)
        reached_target = self.target_position[0:2]
        angle = self.current_pose[5]
        landing_point = [reached_target[0] + np.cos(angle) * self.LANDING_OFFSET,
                         reached_target[1] + np.sin(angle) * self.LANDING_OFFSET]
        self.target_position[0:2] = landing_point
        if verbose_target:
            print("Target image found! Landing at:", landing_point)
    elif self.state == SEARCHING:
        self.target_index += 1
        if self.target_index > len(waypoints) - 1:
            self.target_index = 0
        self.target_position[0:2] = waypoints[self.target_index]
        if verbose_target:
            print("Target point reached! New target:",
                  self.target_position[0:2])
```

درنهایت با کاهش مطلوب ارتفاع، ربات به حالت LANDED می‌رود و موتورها خاموش می‌شوند.

```
    if self.state == LANDING:
        if self.current_pose[2] < self.LAND_ALTITUDE + 0.08:
            self.state = LANDED
    if self.state == LANDED:
        for motor in self.motors:
            motor.setVelocity(0)
        continue
```

شبکه عصبی پیچشی

ابتدا داده های تست و ترین را رو روی کولب میخوانیم. جهت اطمینان و نمایش تصویر ها 5 نمونه از هر لیبل از دسته داده تست را نمایش می دهیم.

یک تابع با نام `sample_images_data` داریم که یک پارامتر به نام `data` میگیرد. این تابع یک لیست خالی به نام `sample_images` ایجاد می‌کند که برای جمع‌آوری تصاویر نمونه استفاده می‌شود.

در ابتدا، این تابع بر روی کلیدهای دیکشنری `labels` که 5 دسته گفته شده در ان تعریف شده است، حلقه می‌زند. سپس برای هر کلید (یعنی هر دسته)، پنج نمونه از داده‌ها را با استفاده از شرط `data["label"] == k` (که به ازای هر دسته معادل True است) انتخاب می‌کند. سپس برای هر یک از این نمونه‌ها، یک حلقه دیگر بر روی اعضای هر نمونه اجرا می‌شود. در هر مرحله، تصویر مربوط به ستون‌های 1 تا آخر از هر نمونه را با استفاده از `np.array(samples.iloc[j, 1:]).reshape(28,28)` به یک آرایه نامپای تبدیل می‌کند(چون ستون 0 شامل لیبل ها است از 1 شروع میکنیم) که ابعاد آن 28x28 است. سپس این تصاویر به لیست `sample_images` اضافه می‌شوند. در نهایت، لیست حاوی تصاویر نمونه بازگردانده می‌شود و در متغیر `test_sample_images` ذخیره می‌شود.

تابع `plot_sample_images` یک لیست از تصاویر نمونه (`data_sample_images`) را گرفته و آن‌ها را در یک نمودار 5x5 نمایش می‌دهد.

ابتدا تعداد ردیف‌ها و ستون‌های نمودار (5 ردیف و 5 ستون) تعیین می‌شود. یک شیء شکل (`fig`) و یک آرایه از محورها (`axes`) با ابعاد مشخص شده ایجاد می‌شوند. با استفاده از دو حلقه `for`، بر روی ردیف‌ها و ستون‌ها حلقه زده می‌شود. ایندکس فعلی بر اساس ردیف و ستون محاسبه می‌شود و اگر این ایندکس کمتر از طول لیست `data_sample_images` باشد، تصویر متناظر با ایندکس در محور مربوطه نمایش داده می‌شود. در غیر این صورت، محور مربوط به آن خاموش می‌شود. در نهایت، با `plt.show()` نمودار نمایش داده می‌شود.

تابع

```python
labels = {0: "T-shirt", 1: "Trouser", 2: "Pullover", 3: "Shoes", 4: "Bag"}
```

```python
def sample_images_data(data):
    sample_images = []
    for k in labels.keys():
        # Get five samples for each category
        samples = data[data["label"] == k].head(5)
        for j, s in enumerate(samples.values):
            img = np.array(samples.iloc[j, 1:]).reshape(28,28)
            sample_images.append(img)
    return sample_images

test_sample_images = sample_images_data(test_data)

def plot_sample_images(data_sample_images, cmap="Greys"):
    num_rows, num_cols = 5, 5
    fig, axes = plt.subplots(num_rows, num_cols, figsize=(8, 10))

    for i in range(num_rows):
        for j in range(num_cols):
            index = i * num_cols + j
            if index < len(data_sample_images):
                axes[i, j].imshow(data_sample_images[index], cmap=cmap)
                axes[i, j].axis('off')
            else:
                axes[i, j].axis('off')

    plt.show()

plot_sample_images(test_sample_images)
```

`data_preprocessing` یک ورودی (`raw_data`) را به عنوان داده‌های خام دریافت می‌کند و آن‌ها را به منظور استفاده در یک مدل شبکه عصبی پیش‌پردازش می‌کند. سپس از این تابع برای پیش‌پردازش داده‌های آموزشی (`train_data`) و داده‌های آزمون (`test_data`) استفاده می‌شود.

نحوه عملکرد این تابع به این شکل است:

ابتدا تعداد دسته‌ها (num_classes) تعیین می‌شود که در اینجا برابر با 5 است.

برچسب‌های خروجی (labels) با استفاده از تابع tf.keras.utils.to_categorical برای تبدیل برچسب‌ها به نمایش one-hot ایجاد می‌شوند.

مقادیر پیکسل (pixel_values) از داده‌های ورودی استخراج می‌شوند.

این مقادیر پیکسل با استفاده از reshape به شکل آرایه‌ای با ابعاد (تعداد تصاویر، ابعاد تصویر در ارتفاع، ابعاد تصویر در عرض، ۱) تبدیل می‌شوند.(28)

در مرحله بعد، مقادیر پیکسل به بازه [0، 1] نرمال‌سازی می‌شوند.

در نهایت، دو خروجی (normalized_images و labels) به ترتیب برابر با داده‌های ورودی پیش‌پردازش‌شده و برچسب‌های one-hot مربوط به تابع باز می‌گردند.

از این تابع برای پیش‌پردازش داده‌های آموزشی و آزمون استفاده می‌شود و نتایج در متغیرهای X_test، y، X و y_test ذخیره می‌شوند.

train_test_split: این تابع از Scikit-Learn برای تقسیم داده‌ها به دو قسمت ترین و اعتبارسنجی استفاده می‌شود. داده‌های ورودی (X و y) به صورت تصادفی به دو مجموعه ترین و اعتبارسنجی تقسیم می‌شوند. آرگومان test_size نسبت تعیین می‌کند که چه نسبتی از داده به عنوان مجموعه اعتبارسنجی اختصاص داده شود. در این پروژه به نسبت 80-20 تقسیم شده است. پارامترrandom_state باعث تعیین یک نقطه شروع ثابت برای تولید اعداد تصادفی می‌شود، که باعث ایجاد تقسیم یکسان برای هر بار اجرا می‌شود. در این پروژه 42 در نظر گرفته شده است.

سپس اندازه مجموعه‌های ترین، اعتبارسنجی و آزمون چاپ می‌شود:

- x_train.shape[0]: تعداد نمونه‌ها در مجموعه آموزش
- x_train.shape[1:4]: ابعاد هر نمونه در مجموعه آموزش (برای داده‌های تصویری این ابعاد شامل ارتفاع، عرض و تعداد کانال‌ها است)
- x_val.shape[0]: تعداد نمونه‌ها در مجموعه اعتبارسنجی
- x_val.shape[1:4]: ابعاد هر نمونه در مجموعه اعتبارسنجی
- X_test.shape[0]: تعداد نمونه‌ها در مجموعه آزمون
- X_test.shape[1:4]: ابعاد هر نمونه در مجموعه آزمون

```python
def data_preprocessing(raw_data):
    num_classes = 5

    # Extract labels and convert to one-hot encoding
    labels = tf.keras.utils.to_categorical(raw_data.label, num_classes)

    # Extract pixel values
    pixel_values = raw_data.values[:, 1:]

    # Reshape pixel values
    num_images = raw_data.shape[0]
    reshaped_images = pixel_values.reshape(num_images, 28, 28, 1)

    # Normalize pixel values to [0, 1]
    normalized_images = reshaped_images / 255.0

    return normalized_images, labels


X, y = data_preprocessing(train_data)
X_test, y_test = data_preprocessing(test_data)
```

```python
x_train, x_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

print("training set size",x_train.shape[0], x_train.shape[1:4])
print("validation set size",x_val.shape[0], x_val.shape[1:4])
print("test set size",X_test.shape[0]," columns:", X_test.shape[1:4])
```

```
training set size 24000 (28, 28, 1)
validation set size 6000 (28, 28, 1)
test set size 5000  columns: (28, 28, 1)
```

برای ساخت مدل از keras که یکی از محبوب ترین API های سطح بالا برای تنسورفلو است استفاده شد.
مقایسه حالت های مختلف تست شده:

Relu and adagrad:

```
cnn = Sequential()
cnn.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
cnn.add(BatchNormalization())

cnn.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
cnn.add(BatchNormalization())
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Dropout(0.25))

cnn.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
cnn.add(BatchNormalization())
cnn.add(Dropout(0.25))

cnn.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
cnn.add(BatchNormalization())
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Dropout(0.25))

cnn.add(Flatten())

cnn.add(Dense(512, activation='relu'))
cnn.add(BatchNormalization())
cnn.add(Dropout(0.5))

cnn.add(Dense(128, activation='relu'))
cnn.add(BatchNormalization())
cnn.add(Dropout(0.5))

cnn.add(Dense(5, activation='softmax'))
```

```
Epoch 86/100
94/94 [==============================] - 2s 23ms/step - loss: 0.1065 - accuracy: 0.9676 - val_loss: 0.0781 - val_accuracy: 0.9767
Epoch 87/100
94/94 [==============================] - 2s 23ms/step - loss: 0.1062 - accuracy: 0.9676 - val_loss: 0.0778 - val_accuracy: 0.9768
Epoch 88/100
94/94 [==============================] - 2s 23ms/step - loss: 0.1059 - accuracy: 0.9672 - val_loss: 0.0775 - val_accuracy: 0.9770
Epoch 89/100
94/94 [==============================] - 2s 23ms/step - loss: 0.1047 - accuracy: 0.9674 - val_loss: 0.0774 - val_accuracy: 0.9768
Epoch 90/100
94/94 [==============================] - 2s 23ms/step - loss: 0.1066 - accuracy: 0.9680 - val_loss: 0.0771 - val_accuracy: 0.9773
Epoch 91/100
94/94 [==============================] - 2s 24ms/step - loss: 0.1101 - accuracy: 0.9659 - val_loss: 0.0768 - val_accuracy: 0.9777
Epoch 92/100
94/94 [==============================] - 2s 25ms/step - loss: 0.1074 - accuracy: 0.9676 - val_loss: 0.0767 - val_accuracy: 0.9775
Epoch 93/100
94/94 [==============================] - 2s 24ms/step - loss: 0.1059 - accuracy: 0.9680 - val_loss: 0.0765 - val_accuracy: 0.9775
Epoch 94/100
94/94 [==============================] - 2s 23ms/step - loss: 0.1035 - accuracy: 0.9681 - val_loss: 0.0763 - val_accuracy: 0.9777
Epoch 95/100
94/94 [==============================] - 2s 23ms/step - loss: 0.1013 - accuracy: 0.9687 - val_loss: 0.0760 - val_accuracy: 0.9778
Epoch 96/100
94/94 [==============================] - 2s 23ms/step - loss: 0.1047 - accuracy: 0.9688 - val_loss: 0.0760 - val_accuracy: 0.9777
Epoch 97/100
94/94 [==============================] - 2s 23ms/step - loss: 0.1029 - accuracy: 0.9684 - val_loss: 0.0758 - val_accuracy: 0.9778
Epoch 98/100
94/94 [==============================] - 2s 25ms/step - loss: 0.1013 - accuracy: 0.9698 - val_loss: 0.0756 - val_accuracy: 0.9780
Epoch 99/100
94/94 [==============================] - 2s 25ms/step - loss: 0.1038 - accuracy: 0.9673 - val_loss: 0.0754 - val_accuracy: 0.9780
Epoch 100/100
94/94 [==============================] - 2s 23ms/step - loss: 0.1056 - accuracy: 0.9671 - val_loss: 0.0752 - val_accuracy: 0.9782
```

```
score = cnn.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.08177521079778671
Test accuracy: 0.9738000035285950
```

Sigmoid and adagrad:

```
cnn = Sequential()
cnn.add(Conv2D(32, kernel_size=(3, 3), activation='sigmoid', input_shape=(28, 28, 1)))
cnn.add(BatchNormalization())

cnn.add(Conv2D(32, kernel_size=(3, 3), activation='sigmoid'))
cnn.add(BatchNormalization())
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Dropout(0.25))

cnn.add(Conv2D(64, kernel_size=(3, 3), activation='sigmoid'))
cnn.add(BatchNormalization())
cnn.add(Dropout(0.25))

cnn.add(Conv2D(128, kernel_size=(3, 3), activation='sigmoid'))
cnn.add(BatchNormalization())
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Dropout(0.25))

cnn.add(Flatten())

cnn.add(Dense(512, activation='sigmoid'))
cnn.add(BatchNormalization())
cnn.add(Dropout(0.5))

cnn.add(Dense(128, activation='sigmoid'))
cnn.add(BatchNormalization())
cnn.add(Dropout(0.5))

cnn.add(Dense(5, activation='softmax'))
```

```
Epoch 86/100
94/94 [==============================] - 2s 24ms/step - loss: 0.1389 - accuracy: 0.9605 - val_loss: 0.1149 - val_accuracy: 0.9712
Epoch 87/100
94/94 [==============================] - 2s 24ms/step - loss: 0.1359 - accuracy: 0.9610 - val_loss: 0.1143 - val_accuracy: 0.9712
Epoch 88/100
94/94 [==============================] - 2s 26ms/step - loss: 0.1333 - accuracy: 0.9629 - val_loss: 0.1142 - val_accuracy: 0.9712
Epoch 89/100
94/94 [==============================] - 3s 27ms/step - loss: 0.1355 - accuracy: 0.9610 - val_loss: 0.1137 - val_accuracy: 0.9717
Epoch 90/100
94/94 [==============================] - 2s 26ms/step - loss: 0.1280 - accuracy: 0.9624 - val_loss: 0.1133 - val_accuracy: 0.9713
Epoch 91/100
94/94 [==============================] - 2s 24ms/step - loss: 0.1349 - accuracy: 0.9615 - val_loss: 0.1136 - val_accuracy: 0.9715
Epoch 92/100
94/94 [==============================] - 2s 24ms/step - loss: 0.1278 - accuracy: 0.9624 - val_loss: 0.1136 - val_accuracy: 0.9713
Epoch 93/100
94/94 [==============================] - 2s 24ms/step - loss: 0.1352 - accuracy: 0.9610 - val_loss: 0.1142 - val_accuracy: 0.9720
Epoch 94/100
94/94 [==============================] - 2s 24ms/step - loss: 0.1297 - accuracy: 0.9625 - val_loss: 0.1130 - val_accuracy: 0.9715
Epoch 95/100
94/94 [==============================] - 3s 28ms/step - loss: 0.1318 - accuracy: 0.9620 - val_loss: 0.1138 - val_accuracy: 0.9718
Epoch 96/100
94/94 [==============================] - 2s 26ms/step - loss: 0.1307 - accuracy: 0.9625 - val_loss: 0.1132 - val_accuracy: 0.9713
Epoch 97/100
94/94 [==============================] - 2s 24ms/step - loss: 0.1272 - accuracy: 0.9628 - val_loss: 0.1125 - val_accuracy: 0.9715
Epoch 98/100
94/94 [==============================] - 2s 24ms/step - loss: 0.1294 - accuracy: 0.9614 - val_loss: 0.1119 - val_accuracy: 0.9713
Epoch 99/100
94/94 [==============================] - 2s 24ms/step - loss: 0.1316 - accuracy: 0.9626 - val_loss: 0.1117 - val_accuracy: 0.9720
Epoch 100/100
94/94 [==============================] - 2s 24ms/step - loss: 0.1235 - accuracy: 0.9638 - val_loss: 0.1115 - val_accuracy: 0.9720
```

```
score = cnn.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.1118319183588028
Test accuracy: 0.9729999899864197
```

Relu and adam: بهترین نتیجه

```
Epoch 91/100
94/94 [==============================] - 2s 25ms/step - loss: 0.0027 - accuracy: 0.9992 - val_loss: 0.1008 - val_accuracy: 0.9865
Epoch 92/100
94/94 [==============================] - 2s 25ms/step - loss: 0.0030 - accuracy: 0.9988 - val_loss: 0.0923 - val_accuracy: 0.9870
Epoch 93/100
94/94 [==============================] - 2s 23ms/step - loss: 0.0028 - accuracy: 0.9991 - val_loss: 0.0975 - val_accuracy: 0.9870
Epoch 94/100
94/94 [==============================] - 2s 23ms/step - loss: 0.0031 - accuracy: 0.9991 - val_loss: 0.0838 - val_accuracy: 0.9875
Epoch 95/100
94/94 [==============================] - 2s 23ms/step - loss: 0.0022 - accuracy: 0.9992 - val_loss: 0.0884 - val_accuracy: 0.9875
Epoch 96/100
94/94 [==============================] - 2s 23ms/step - loss: 0.0032 - accuracy: 0.9988 - val_loss: 0.0949 - val_accuracy: 0.9853
Epoch 97/100
94/94 [==============================] - 2s 24ms/step - loss: 0.0036 - accuracy: 0.9987 - val_loss: 0.0975 - val_accuracy: 0.9873
Epoch 98/100
94/94 [==============================] - 2s 25ms/step - loss: 0.0026 - accuracy: 0.9990 - val_loss: 0.0869 - val_accuracy: 0.9863
Epoch 99/100
94/94 [==============================] - 2s 24ms/step - loss: 0.0024 - accuracy: 0.9994 - val_loss: 0.1028 - val_accuracy: 0.9863
Epoch 100/100
94/94 [==============================] - 2s 23ms/step - loss: 0.0023 - accuracy: 0.9994 - val_loss: 0.0930 - val_accuracy: 0.9863
```

```
score = cnn.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

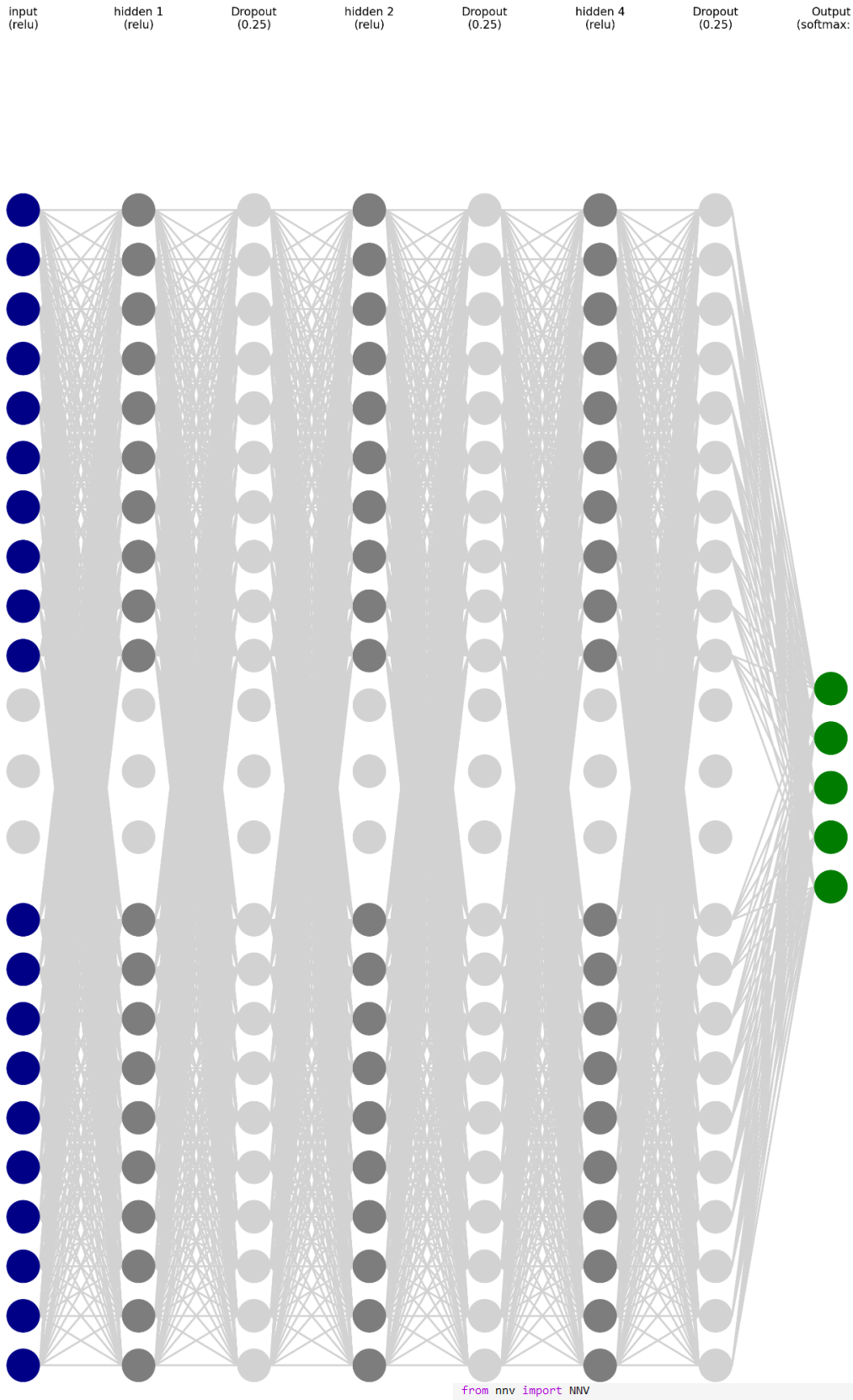
```
Test loss: 0.0689203292131424
Test accuracy: 0.9882000088691711
```

Sigmoid and adam:

```
94/94 [==============================] - 2s 25ms/step - loss: 0.0050 - accuracy: 0.9984 - val_loss: 0.0776 - val_accuracy: 0.9858
Epoch 89/100
94/94 [==============================] - 2s 24ms/step - loss: 0.0054 - accuracy: 0.9981 - val_loss: 0.0792 - val_accuracy: 0.9847
Epoch 90/100
94/94 [==============================] - 2s 24ms/step - loss: 0.0073 - accuracy: 0.9978 - val_loss: 0.0849 - val_accuracy: 0.9847
Epoch 91/100
94/94 [==============================] - 2s 24ms/step - loss: 0.0080 - accuracy: 0.9969 - val_loss: 0.0877 - val_accuracy: 0.9828
Epoch 92/100
94/94 [==============================] - 2s 26ms/step - loss: 0.0080 - accuracy: 0.9973 - val_loss: 0.0868 - val_accuracy: 0.9848
Epoch 93/100
94/94 [==============================] - 3s 28ms/step - loss: 0.0064 - accuracy: 0.9979 - val_loss: 0.0815 - val_accuracy: 0.9855
Epoch 94/100
94/94 [==============================] - 2s 25ms/step - loss: 0.0059 - accuracy: 0.9981 - val_loss: 0.0915 - val_accuracy: 0.9843
Epoch 95/100
94/94 [==============================] - 2s 24ms/step - loss: 0.0063 - accuracy: 0.9977 - val_loss: 0.0832 - val_accuracy: 0.9853
Epoch 96/100
94/94 [==============================] - 2s 24ms/step - loss: 0.0074 - accuracy: 0.9977 - val_loss: 0.0911 - val_accuracy: 0.9820
Epoch 97/100
94/94 [==============================] - 2s 24ms/step - loss: 0.0050 - accuracy: 0.9983 - val_loss: 0.0842 - val_accuracy: 0.9840
Epoch 98/100
94/94 [==============================] - 2s 25ms/step - loss: 0.0074 - accuracy: 0.9976 - val_loss: 0.0953 - val_accuracy: 0.9833
Epoch 99/100
94/94 [==============================] - 3s 29ms/step - loss: 0.0066 - accuracy: 0.9978 - val_loss: 0.0849 - val_accuracy: 0.9857
Epoch 100/100
94/94 [==============================] - 2s 26ms/step - loss: 0.0085 - accuracy: 0.9977 - val_loss: 0.0828 - val_accuracy: 0.9850
```

```
score = cnn.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

مدل شبکه عصبی:



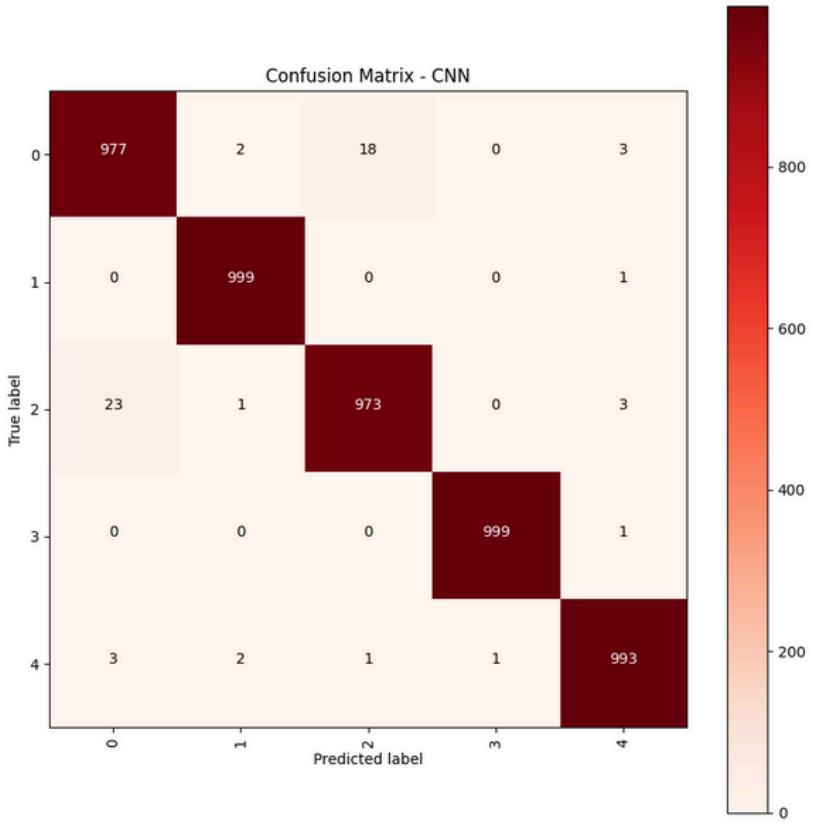

```
from nnv import NNV
plt.rcParams["figure.figsize"] = 100,50

layersList = [
    {"title":"input\n(relu)", "units": 784, "color": "darkBlue"},
    {"title":"hidden 1\n(relu)", "units": 32},
    {"title":"Dropout\n(0.25)", "units": 32,"color":"lightGray"},
    {"title":"hidden 2\n(relu)",  "units": 64},
    {"title":"Dropout\n(0.25)", "units": 64,"color":"lightGray"},
    {"title":"hidden 4\n(relu)",  "units": 128},
    {"title":"Dropout\n(0.25)", "units": 128,"color":"lightGray"},
    {"title":"Output\n(softmax: 5)", "units": 5,"color": "Green"},
]

NNV(layersList, spacing_layer=5, max_num_nodes_visible=20, node_radius=1, font_size=24).render()
```

یک مدل سکوئنشال ایجاد می‌شود.
لایه کانولوشنال با 32 فیلتر، ابعاد کرنل 3x3، تابع فعال‌سازی ReLU و ابعاد ورودی (input_shape) 28x28x1 اضافه می‌شود.
یک لایه Batch Normalization پس از لایه کانولوشنال اضافه می‌شود. این لایه به منظور استقرار توزیع ورودی‌ها و بهبود مشکلات آموزشی مورد استفاده قرار می‌گیرد.
یک لایه کانولوشنال دیگر با 32 فیلتر، ابعاد کرنل 3x3 و تابع فعال‌سازی ReLU اضافه می‌شود.
یک لایه Batch Normalization دیگر اضافه می‌شود.
یک لایه MaxPooling با ابعاد 2x2 اضافه می‌شود.
یک لایه Dropout با نرخ Dropout برابر با 0.25 اضافه می‌شود. Dropout به منظور جلوگیری از برازش بیش از حد مدل به داده‌های آموزشی مورد استفاده قرار می‌گیرد.
یک لایه Flatten اضافه می‌شود تا از لایه‌های کانولوشنال به لایه‌های کاملاً متصل منتقل شویم.
یک لایه Dense با 512 نرون، تابع فعال‌سازی ReLU اضافه می‌شود.
یک لایه Batch Normalization و یک لایه Dropout با نرخ Dropout برابر با 0.5 اضافه می‌شود.
یک لایه Dense با 5 نرون و تابع فعال‌سازی softmax برای دسته‌بندی اضافه می‌شود.
در مجموع 4 لایه کانولوشنی در این شبکه عصبی اعمال شده است.

Confusion Matrix:

همانطور که دیده می شود عملکرد مدل بسیار خوب است و در تصاویر تیشرت و پلیور دچار خطا میشود که کاملا قابل پیش بینی است زیرا در واقعیت هم ساختار این دو شبیه به هم است.

```
157/157 [==============================] - 1s 2ms/step
```




```python
from sklearn.metrics import confusion_matrix
from itertools import product

#Create Multiclass Confusion Matrix

preds = cnn.predict(X_test)
cm = confusion_matrix(np.argmax(y_test,axis=1), np.argmax(preds,axis=1))

plt.figure(figsize=(8,8))
plt.imshow(cm,cmap=plt.cm.Reds)
plt.title('Confusion Matrix - CNN')
plt.colorbar()
plt.xticks(np.arange(5), labels, rotation=90)
plt.yticks(np.arange(5), labels)

for i, j in product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, cm[i, j],
    horizontalalignment="center",
    color="white" if cm[i, j] > 500 else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label');
```

برای تست تصویرهای دیگر با مدل آن را سیو میکنیم:

```python
# save model
cnn.save('model_100.h5')
```

یک تابع برای اصلاح جهت تصویر قرار میدهیم:

```
from PIL import Image
import math

def rotate_image(input_path, output_path, angle):
    # Open the image
    img = Image.open(input_path)

    # Rotate the image by the specified angle
    rotated_img = img.rotate(angle, resample=Image.BICUBIC, expand=True)

    # Save the rotated image
    rotated_img.save(output_path)

# Example usage
input_image_path = 'image.jpg'  # the path to your input image
output_image_path = 'rotated_image.jpg'
rotation_angle = 90  # the desired rotation angle in degrees

rotate_image(input_image_path, output_image_path, rotation_angle)
```

این تابع درجه لازم برای چرخش را میگیرد و تصویر چرخیده را سیو میکند.

در قسمت بعد باید تصویر صحیح را به مدل بدهیم تا دسته بندی را انجام دهد
یک تابع برای تنظیم سایز 28 در 28 و تبدیل به ارایه کردن تصویر و grayscale لازم داریم و یک تابع برای دادن عکس به مدل. مدل را لود میکنیم و تصویر آماده را به آن می دهیم.

```
def load_and_prepare_image(filename, target_size=(28, 28)):
    # Load the image
    img = load_img(filename, grayscale=True, target_size=target_size)
    # Convert to array
    img_array = img_to_array(img)
    # Reshape into a single sample with 1 channel
    img_array = np.expand_dims(img_array, axis=0)
    # Prepare pixel data
    img_array = img_array.astype('float32') / 255.0
    return img_array

def predict_class(model, img):
    # Predict the class
    predictions = model.predict(img)
    predicted_class_index = np.argmax(predictions[0])
    return predicted_class_index

def tester():
    # Specify the image file path
    image_file = 'rotated_image.jpg'

    # Load and prepare the image
    img = load_and_prepare_image(image_file)

    # Load the model
    model = load_model('model_100.h5')

    # Predict the class
    predicted_class = predict_class(model, img)

    # Print the predicted class
    print(f"Predicted Class Index: {predicted_class}")

# Entry point, run the example
tester()
```

برای تست مدل تصویر ها را با روتیشن به مدل میدهیم:

```python
def tester():
    # Specify the image file path
    image_file = 'rotated_image.jpg'

    # Load and prepare the image
    img = load_and_prepare_image(image_file)

    # Load the model
    model = load_model('model_100.h5')

    # Predict the class
    predicted_class = predict_class(model, img)

    # Print the predicted class
    print(f"Predicted Class Index: {predicted_class}")

# Entry point, run the example
tester()
```

```
1/1 [==============================] - 0s 277ms/step
Predicted Class Index: 0
```

rotated_image.jpg

به درستی تیشرت دسته بندی میکند.

```python
def tester():
    # Specify the image file path
    image_file = 'rotated_image.jpg'

    # Load and prepare the image
    img = load_and_prepare_image(image_file)

    # Load the model
    model = load_model('model_100.h5')

    # Predict the class
    predicted_class = predict_class(model, img)

    # Print the predicted class
    print(f"Predicted Class Index: {predicted_class}")

# Entry point, run the example
tester()
```

```
1/1 [==============================] - 0s 251ms/step
Predicted Class Index: 1
```

rotated_image.jpg

به درستی شلوار دسته بندی میکند.

```python
def tester():
    # Specify the image file path
    image_file = 'rotated_image.jpg'

    # Load and prepare the image
    img = load_and_prepare_image(image_file)

    # Load the model
    model = load_model('model_100.h5')

    # Predict the class
    predicted_class = predict_class(model, img)

    # Print the predicted class
    print(f"Predicted Class Index: {predicted_class}")

# Entry point, run the example
tester()
```

```
1/1 [==============================] - 0s 170ms/step
Predicted Class Index: 2
```

rotated_image.jpg

به درستی پلیور دسته بندی میکند.

```python
def tester():
    # Specify the image file path
    image_file = 'rotated_image.jpg'

    # Load and prepare the image
    img = load_and_prepare_image(image_file)

    # Load the model
    model = load_model('model_100.h5')

    # Predict the class
    predicted_class = predict_class(model, img)

    # Print the predicted class
    print(f"Predicted Class Index: {predicted_class}")

# Entry point, run the example
tester()
```

```
1/1 [==============================] - 0s 170ms/step
Predicted Class Index: 3
```

rotated_image.jpg ×

به درستی کفش دسته بندی میکند.

```python
def tester():
    # Specify the image file path
    image_file = 'rotated_image.jpg'

    # Load and prepare the image
    img = load_and_prepare_image(image_file)

    # Load the model
    model = load_model('model_100.h5')

    # Predict the class
    predicted_class = predict_class(model, img)

    # Print the predicted class
    print(f"Predicted Class Index: {predicted_class}")

# Entry point, run the example
tester()
```

```
1/1 [==============================] - 0s 171ms/step
Predicted Class Index: 4
```

rotated_image.jpg ×

به درستی کیف دسته بندی میکند.

اتصال دو بخش

مدل ساخته شده در مرحله قبل در یک فایل خروجی گرفته می‌شود و در کنار فعال‌سازی سایر سنسورهای ربات، خوانده و آماده استفاده می‌شود.
قبل از دادن تصویر گرفته شده دوربین به شبکه عصبی، نیاز به پیش‌پردازش آن داریم. ابتدا تصویر را با توجه به زاویه ربات می‌چرخانیم تا تصویر روی جعبه صاف باشد. برای اینکه تصویر نهایی ناشی از این چرخش، حاشیه سیاه نداشته باشد، بزرگترین مستطیل مرکزی موجود در آن را crop می‌کنیم و بقیه پیکسل‌ها را حذف می‌کنیم.

```
image_height, image_width = image.shape[0:2]
img = Image.fromarray(image.astype(np.uint8))
rotated_img = np.array(img.rotate(angle, resample=Image.BICUBIC, expand=True))
return np.array(crop_around_center(
    rotated_img,
    *largest_rotated_rect(
        image_width,
        image_height,
        math.radians(angle)
    )
))
```

سپس در تصویر چرخانده شده، به دنبال پیکسل‌های جعبه سیاه می‌گردیم. هر پیکسلی که روشنایی‌اش از حد مشخصی کمتر باشد را جزو جعبه حساب می‌کنیم. کمترین و بیشترین x این پیکسل‌ها و همچنین کمترین و بیشترین y این پیکسل ها را پیدا می‌کنیم و تصویر را طبق این ۴ مقدار crop می‌کنیم.

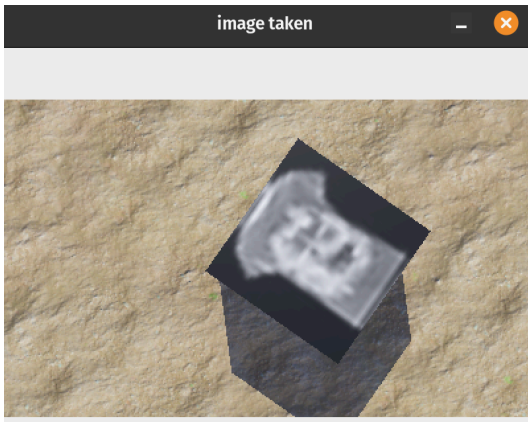
```
def crop(image):
    grey = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    x_min, x_max = grey.shape[0]-1, 0
    y_min, y_max = grey.shape[1]-1, 0
    for i in range(grey.shape[0]):
        for j in range(grey.shape[1]):
            if grey[i, j] < 48:
                x_min, x_max = min(x_min, i), max(x_max, i)
                y_min, y_max = min(y_min, j), max(y_max, j)

    output_image = image[x_min:x_max, y_min:y_max]
    return output_image
```

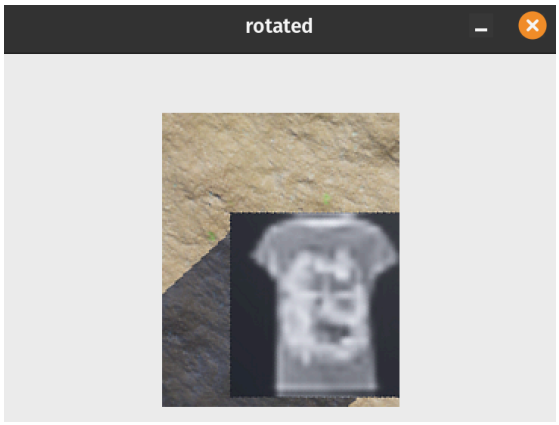
سپس تصویر را بسیاه و سفید کرده، به ابعاد ورودی شبکه عصبی در می‌آوریم و مقادیر را normalize می‌کنیم.

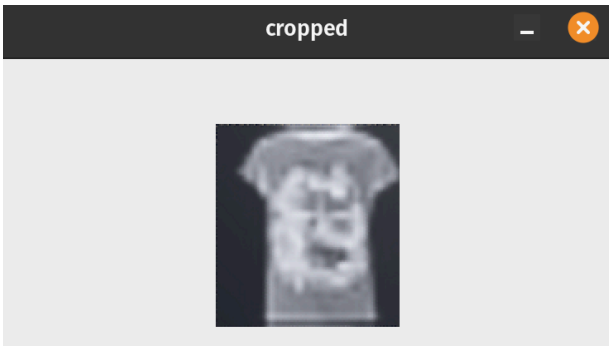
```
def prepare_image(img_array, target_size=(28, 28)):
    # Convert to PIL Image
    img = array_to_img(img_array)
    # Resize the image
    img = img.convert('L').resize(target_size)
    # Convert to array
    img_array = img_to_array(img)
    # Reshape into a single sample with 1 channel
    img_array = np.expand_dims(img_array, axis=0)
    # Prepare pixel data
    img_array = img_array.astype('float32') / 255.0
    return img_array
```

در نهایت آن را به شبکه عصبی می‌دهیم و کلاسی که بیشترین احتمال را در خروجی دارد با استفاده از argmax پیدا می‌کنیم.

```python
def predict_class(model, img, angle_degrees):
    rotated_img = rotate_image(img, angle_degrees)
    show(rotated_img, title: 'rotated')
    cropped_img = crop(rotated_img)
    show(cropped_img, title: 'cropped')
    model_input = prepare_image(cropped_img)

    predictions = model.predict(model_input)
    predicted_class_index = np.argmax(predictions[0])
    return predicted_class_index
```







در کد کنترلر ربات، تابعی به نام check_target_image داریم که یک تصویر با استفاده از دوربین می‌گیرد و آن را به تابع predict_class می‌دهد و خروجی را با مقدار از پیش تعیین شده target_image_class مقایسه می‌کند. زاویه‌ای که عکس باید چرخانده شود تا صاف شود، ۱۸۰ درجه بیشتر از زاویه کنونی ربات است.

```python
def check_target_image(self):
    img = self.take_picture()
    img_bgr = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    cv2.imshow( winname: 'image taken', img_bgr)
    # cv2.imwrite(f'img{self.camera_counter}.png', img_bgr)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    self.camera_counter += 1

    prediction = cnn.predict_class(self.cnn_model, img_bgr, math.degrees(self.current_pose[5] + np.pi))
    return prediction == self.target_image_class
```