```
import numpy as np
import tensorflow as tf
from keras.datasets import mnist
from tensorflow.keras.models import Sequential
#from ann_visualizer.visualize import ann_viz
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt


from keras.callbacks import CSVLogger


csv_logger = CSVLogger("model_history_log.csv", append=True)


(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
11490434/11490434 [==============================] - 0s 0us/step
```

```
#loding the images
#train_images = mnist.train_images()
#train_labels = mnist.train_labels()
print(len(train_labels))
#test_images = mnist.test_images()
#test_labels = mnist.test_labels()
print(len(test_labels))

# Normalize the images. Pixel values are between 0-255 in image learning it is
# good practice to normalize your data to a smaller range like between 0 and 1.
train_images = (train_images / 255) - 0.5
#print(train_images)
test_images = (test_images / 255) - 0.5
#print(test_images)
```

```
60000
10000
```

```
#lets give the hyper parameters
num_filters = 1
filter_size = 9
pool_size = 2

# Model is being trained on 1875 batches of 32 images each, not 1875 images. 1875*
# Build the model.
model = Sequential([
  Conv2D(num_filters, filter_size, input_shape=(28, 28, 1)),
  MaxPooling2D(pool_size=pool_size),
  Flatten(),
  Dense(10, activation='softmax'),
])
```

```
# Compile the model.
model.compile(
  'adam',
  loss='categorical_crossentropy',
  metrics=['accuracy'],
)
```

```
tf.keras.callbacks.EarlyStopping(
      monitor="loss",
      min_delta=0,
      patience=0,
      verbose=0,
      mode="auto",
      baseline=None,
      restore_best_weights=False,
    )
```

```
<keras.src.callbacks.EarlyStopping at 0x7f90f9d00fa0>
```

```
callback_1 = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=2)
```

```
# Train the model.
history = model.fit(
  train_images,
  to_categorical(train_labels),
  epochs=10,
  validation_data=(test_images, to_categorical(test_labels)),
  callbacks=[csv_logger, callback_1],
)
```

```
Epoch 1/10
1875/1875 [==============================] - 28s 14ms/step - loss: 0.5639 - ac
Epoch 2/10
1875/1875 [==============================] - 26s 14ms/step - loss: 0.3027 - ac
Epoch 3/10
1875/1875 [==============================] - 26s 14ms/step - loss: 0.2734 - ac
Epoch 4/10
1875/1875 [==============================] - 27s 15ms/step - loss: 0.2550 - ac
Epoch 5/10
1875/1875 [==============================] - 25s 14ms/step - loss: 0.2438 - ac
Epoch 6/10
1875/1875 [==============================] - 25s 14ms/step - loss: 0.2349 - ac
Epoch 7/10
1875/1875 [==============================] - 25s 14ms/step - loss: 0.2292 - ac
Epoch 8/10
1875/1875 [==============================] - 27s 14ms/step - loss: 0.2241 - ac
Epoch 9/10
1875/1875 [==============================] - 25s 14ms/step - loss: 0.2200 - ac
Epoch 10/10
1875/1875 [==============================] - 26s 14ms/step - loss: 0.2172 - ac
```

```
model.summary()
```

```
Model: "sequential_3"
```

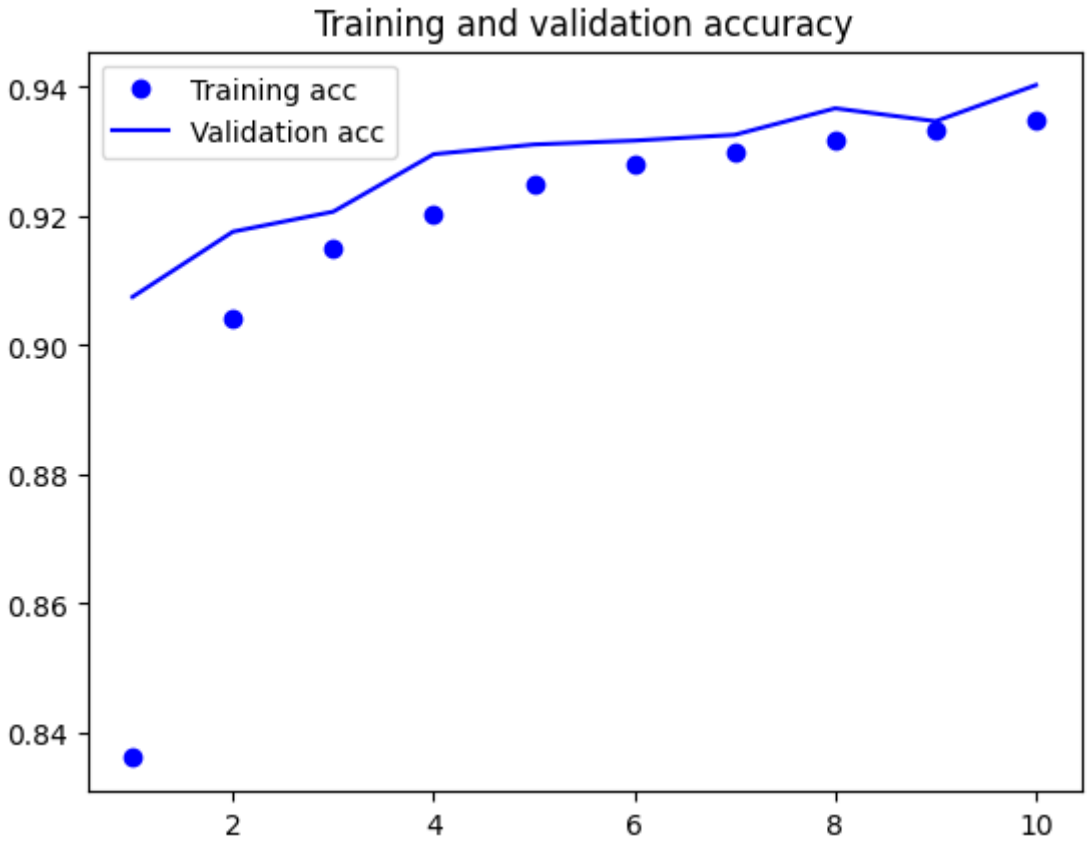
```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 20, 20, 1)         82

 max_pooling2d_3 (MaxPoolin  (None, 10, 10, 1)         0
 g2D)

 flatten_3 (Flatten)         (None, 100)               0

 dense_3 (Dense)             (None, 10)                1010

=================================================================
Total params: 1092 (4.27 KB)
Trainable params: 1092 (4.27 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

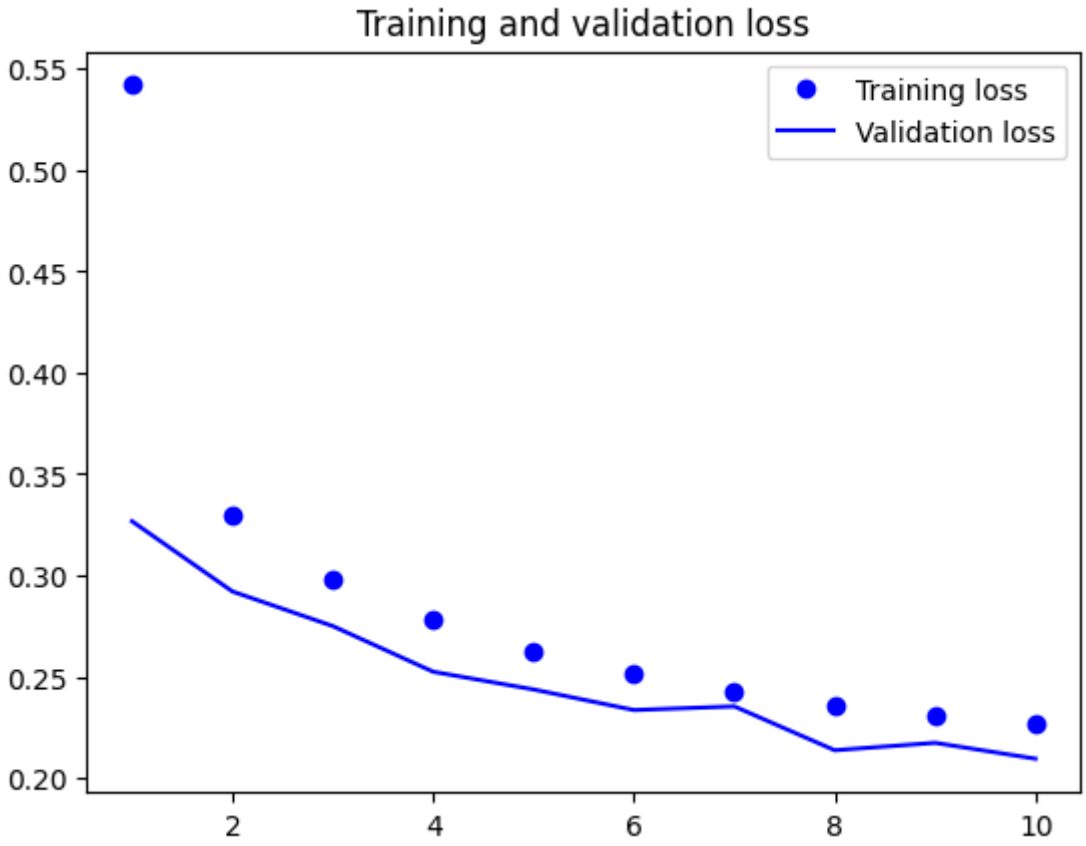
```
#Save the model:
model.save_weights('cnn.h5')

# Load the model's saved weights.
model.load_weights('cnn.h5')


# Predict on the first 10 test images.
predictions = model.predict(test_images[:10])

# Print our model's predictions.
print(np.argmax(predictions, axis=1)) # [7, 2, 1, 0, 4]

# Check our predictions against the ground truths.
print(test_labels[:10]) # [7, 2, 1, 0, 4]
```

```
1/1 [==============================] - 0s 61ms/step
[7 2 1 0 4 1 4 9 5 9]
[7 2 1 0 4 1 4 9 5 9]
```

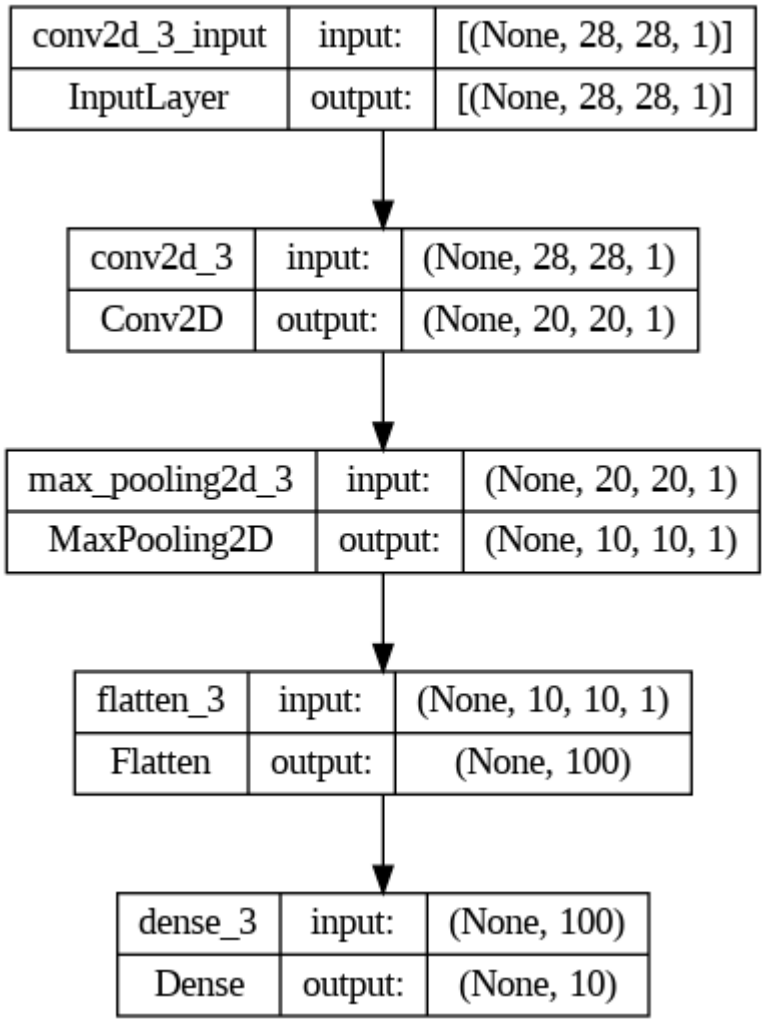
```
#Callback records events into a History object.
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']


#The range of epochs, from 1 to the total number of epochs, is often used to plot
#the range function is used to generate a list of integers from 1 to the length of
# This is because the accuracy list is typically recorded at the end of each epoch
epochs = range(1, len(accuracy) + 1)

plt.plot(epochs, accuracy, 'bo', label='Training acc')
plt.plot(epochs, val_accuracy, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()
```

```
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





```
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=Tru
```




```
! git clone https://github.com/samson6460/tf_keras_gradcamplusplus
```

```
Cloning into 'tf_keras_gradcamplusplus'...
remote: Enumerating objects: 105, done.
remote: Counting objects: 100% (105/105), done.
remote: Compressing objects: 100% (76/76), done.
remote: Total 105 (delta 39), reused 82 (delta 22), pack-reused 0
Receiving objects: 100% (105/105), 13.66 MiB | 30.15 MiB/s, done.
Resolving deltas: 100% (39/39), done.
```

```
%cd tf_keras_gradcamplusplus
```

```
/content/tf_keras_gradcamplusplus
```

```
! pip install -r requirement.txt
```

```
Requirement already satisfied: tensorflow>=2.0.0 in /usr/local/lib/python3.10/
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dis
```

```
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/loc
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.1
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/pyth
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.1
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/lo
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.1
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/pytho
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/loca
Requirement already satisfied: keras<2.16,>=2.15.0 in /usr/local/lib/python3.1
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/di
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.1
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/l
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/pyth
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyth
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/di
```

Since the gradcam code is hardwired for a specific layer, changing the layer argument below in the cell directly.

```
# Copyright 2020 Samson Woof

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at

#     http://www.apache.org/licenses/LICENSE-2.0

# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================
```

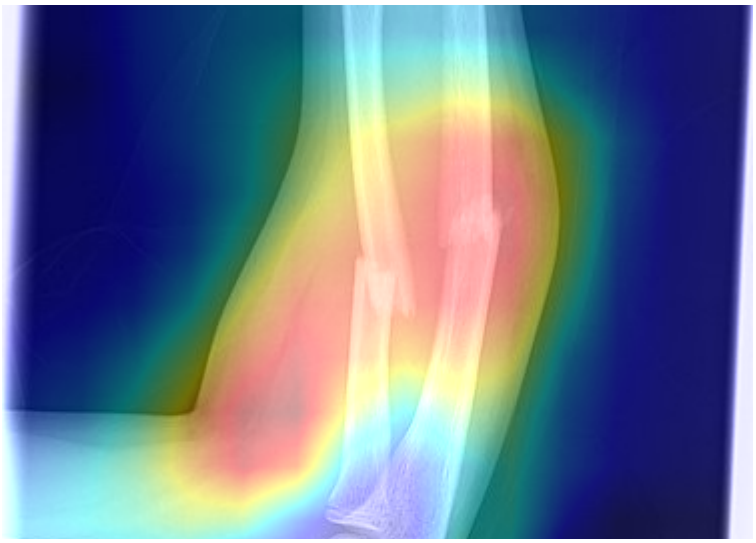
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from tensorflow.keras import Model


def grad_cam(model, img,
             layer_name="conv2d_3", label_name=None,
             category_id=None):
    """Get a heatmap by Grad-CAM.

    Args:
        model: A model object, build from tf.keras 2.X.
        img: An image ndarray.
        layer_name: A string, layer name in model.
        label_name: A list or None,
            show the label name by assign this argument,
            it should be a list of all label names.
        category_id: An integer, index of the class.
            Default is the category with the highest score in the prediction.

    Return:
        A heatmap ndarray(without color).
    """
    img_tensor = np.expand_dims(img, axis=0)

    conv_layer = model.get_layer(layer_name)
    heatmap_model = Model([model.inputs], [conv_layer.output, model.output])

    with tf.GradientTape() as gtape:
        conv_output, predictions = heatmap_model(img_tensor)
        if category_id is None:
            category_id = np.argmax(predictions[0])
        if label_name is not None:
            print(label_name[category_id])
        output = predictions[:, category_id]
        grads = gtape.gradient(output, conv_output)
        pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    heatmap = tf.reduce_mean(tf.multiply(pooled_grads, conv_output), axis=-1)
    heatmap = np.maximum(heatmap, 0)
    max_heat = np.max(heatmap)
    if max_heat == 0:
        max_heat = 1e-10
    heatmap /= max_heat

    return np.squeeze(heatmap)

def grad_cam_plus(model, img,
                  layer_name="conv2d_3", label_name=None,
                  category_id=None):
    """Get a heatmap by Grad-CAM++.
```

```
    Args:
        model: A model object, build from tf.keras 2.X.
        img: An image ndarray.
        layer_name: A string, layer name in model.
        label_name: A list or None,
            show the label name by assign this argument,
            it should be a list of all label names.
        category_id: An integer, index of the class.
            Default is the category with the highest score in the prediction.

    Return:
        A heatmap ndarray(without color).
    """
    img_tensor = np.expand_dims(img, axis=0)

    conv_layer = model.get_layer(layer_name)
    heatmap_model = Model([model.inputs], [conv_layer.output, model.output])

    with tf.GradientTape() as gtape1:
        with tf.GradientTape() as gtape2:
            with tf.GradientTape() as gtape3:
                conv_output, predictions = heatmap_model(img_tensor)
                if category_id is None:
                    category_id = np.argmax(predictions[0])
                    print(category_id)
                if label_name is not None:
                    print(label_name[category_id])
                output = predictions[:, category_id]
                conv_first_grad = gtape3.gradient(output, conv_output)
            conv_second_grad = gtape2.gradient(conv_first_grad, conv_output)
        conv_third_grad = gtape1.gradient(conv_second_grad, conv_output)

    global_sum = np.sum(conv_output, axis=(0, 1, 2))

    alpha_num = conv_second_grad[0]
    alpha_denom = conv_second_grad[0]*2.0 + conv_third_grad[0]*global_sum
    alpha_denom = np.where(alpha_denom != 0.0, alpha_denom, 1e-10)

    alphas = alpha_num/alpha_denom
    alpha_normalization_constant = np.sum(alphas, axis=(0,1))
    alphas /= alpha_normalization_constant

    weights = np.maximum(conv_first_grad[0], 0.0)

    deep_linearization_weights = np.sum(weights*alphas, axis=(0,1))
    grad_cam_map = np.sum(deep_linearization_weights*conv_output[0], axis=2)

    heatmap = np.maximum(grad_cam_map, 0)
    max_heat = np.max(heatmap)
    if max_heat == 0:
        max_heat = 1e-10
    heatmap /= max_heat

    return heatmap
```

The VGG model already available and trained for medical dataset.

```
from utils import vgg16_mura_model, preprocess_image, show_imgwithheat
# from gradcam import grad_cam, grad_cam_plus

# %% load the model
model.summary()

# %%
# %% img_path = 'images/4320878114_30a836d428_z.jpg'
# %% img = preprocess_image(img_path)

# %% result of grad cam
heatmap = grad_cam(model, test,
                   label_name = ['WRIST', 'ELBOW', 'SHOULDER'],
                   #category_id = 0,
                   )
show_imgwithheat(img_path, heatmap)

# %% result of grad cam++
heatmap_plus = grad_cam_plus(model, img)
show_imgwithheat(img_path, heatmap_plus)
```

```
Downloading data from https://github.com/samson6460/tf_keras_gradcamplusplus/r
134511384/134511384 [==============================] - 2s 0us/step
WARNING:tensorflow:No training configuration found in the save file, so the mc
Model: "model_37"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16_input (InputLayer)    [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 global_average_pooling2d_1  (None, 512)               0
  (GlobalAveragePooling2D)

 dense_6 (Dense)             (None, 4096)              2101248

 dense_7 (Dense)             (None, 4096)              16781312

 dense_8 (Dense)             (None, 3)                 12291

=================================================================
Total params: 33609539 (128.21 MB)
Trainable params: 33609539 (128.21 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
ELBOW
```



```
test=test_images[:1]
test.shape
```

```
(1, 28, 28)
```

```
# %% load the model
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 20, 20, 1)         82

 max_pooling2d_3 (MaxPoolin  (None, 10, 10, 1)         0
 g2D)

 flatten_3 (Flatten)         (None, 100)               0

 dense_3 (Dense)             (None, 10)                1010

=================================================================
Total params: 1092 (4.27 KB)
Trainable params: 1092 (4.27 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# %% result of grad cam
heatmap = grad_cam(model, test.transpose([1,2,0]),
                   label_name = range(10),
                   #category_id = 0,
                  )
```

```
7
```

```
import cv2
cv2.imwrite("test.png",test.transpose([1,2,0]))
```