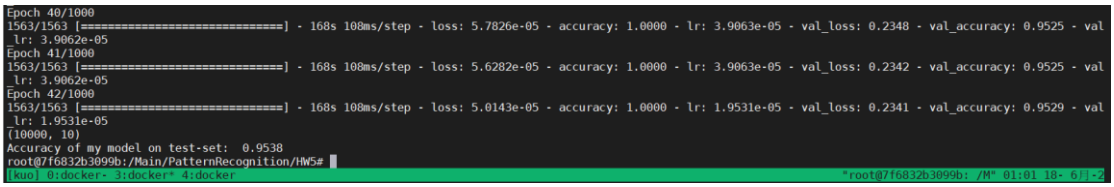
Pattern Recognition HW5

309551101 郭育麟

1. (100%) Show your accuracy of your model on the provided test data by screenshot the results of your code and paste them on your report

Result



Code

In this homework , I use pretrained model ResNet50 and upsample the image size from 32*32*3 to 224*224*3 and replace the final fully connected layer to classify 10 classes . And I also implemented ResNet18 and ResNet34 , but the pretrained ResNet50 has the best performance , So I only paste the screenshot of ResNet50 .

```python
def resnet_50(input_shape=(32, 32, 3), nclass=10):
    model = ResNet50(include_top=False, weights='imagenet', input_shape=input_shape)
    x = model.layers[-1].output
    x = GlobalAvgPool2D()(x)
    x = Dense(nclass, activation='softmax')(x)
    model = Model(model.input, outputs=x)
    return model
```

This function could return pretrained ResNet50 with predict 10 classes

```python
def insert_layer_nonseq(model, layer_regex, insert_layer_factory,
                        insert_layer_name="upsample", position='before'):
    # Auxiliary dictionary to describe the network graph
    network_dict = {'input_layers_of': {}, 'new_output_tensor_of': {}}
    # Set the input layers of each layer
    for layer in model.layers:
        for node in layer.outbound_nodes:
            layer_name = node.outbound_layer.name
            if layer_name not in network_dict['input_layers_of']:
                network_dict['input_layers_of'].update(
                        {layer_name: [layer.name]})
            else:
                network_dict['input_layers_of'][layer_name].append(layer.name)
    # Set the output tensor of the input layer
    network_dict['new_output_tensor_of'].update(
            {model.layers[0].name: model.input})
    # Iterate over all layers after the input
    for layer in model.layers[1:]:
        # Determine input tensors
        layer_input = [network_dict['new_output_tensor_of'][layer_aux]
                for layer_aux in network_dict['input_layers_of'][layer.name]]
        if len(layer_input) == 1:
            layer_input = layer_input[0]
        # Insert layer if name matches the regular expression

        if re.match(layer_regex, layer.name):
            x = model.get_layer("input_1").output
            if position == 'replace':
                x = layer_input
            elif position == 'after':
                x = layer(layer_input)
            elif position == 'before':
                pass
            else:
```

```
                new_layer._name = '{}_{}'.format(layer.name,
                                                 new_layer.name)
            x = new_layer(x)
            print('Layer {} inserted after layer {}'.format(new_layer.name,
                                                            layer.name))
            if position == 'before':
                x = layer(x)
        else:
            x = layer(layer_input)
        # Set new output tensor (the original one, or the one of the inserted
        # layer)
        network_dict['new_output_tensor_of'].update({layer.name: x})
    return Model(inputs=model.inputs, outputs=x)
```

This function could insert upsample layer in the middle of the ResNet50

```
def UpSampling():
    return UpSampling2D(size=(7, 7), interpolation="bilinear")

def get_lr_metric(optimizer):  # printing the value of the learning rate
    def lr(y_true, y_pred):
        return optimizer.lr
    return lr
```

Main :

```
model = resnet_50()
model = insert_layer_nonseq(model, '.*conv1_pad.*', UpSampling)

plot_model(model, show_shapes=True, show_layer_names=True,to_file='model.png')
save_dir=f"./Saved_Model/{args.model_name}"
filepath = "model.h5"
checkpoint = ModelCheckpoint(os.path.join(save_dir, filepath), monitor='val_accuracy', verbose=0, save_best_only=True,mode='auto')
Earlystop = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', mode='auto',patience=15)
Learningrate = tf.keras.callbacks.ReduceLROnPlateau(monitor='loss', factor=0.5, patience=2, verbose=0, mode='auto', epsilon=0.0001, cooldown=0, min_lr=1e-6)
callbacks_list = [checkpoint,history,Learningrate,Earlystop]
# initiate SGD optimizer
opt = keras.optimizers.SGD(learning_rate=1e-2, decay=1e-6, momentum=0.9)
lr_metric = get_lr_metric(opt)
# Compile the model with loss function and optimizer
# model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])
model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy', lr_metric])




# Fit the data into model
# model.fit(datagen.flow(x_train, y_train, batch_size=args.bs),batch_size=args.bs, epochs=args.epoch, validation_data=(x_test, y_test), callbacks = callbacks_l
model.fit(x_train, y_train, batch_size=args.bs, epochs=args.epoch, validation_data=(x_test, y_test), callbacks = callbacks_list, verbose = 1, shuffle=True)
```