# 인공지능 응용/게임프로그래밍 패턴 기말과제

## 딥러닝 기말과제 구현설명서

홍익대학교 게임학부 게임소프트웨어학과

B793058 김지형

Kimsu802222@naver.com

## Implementation Manual on Deep Learning Code Refactoring

B793058 Kim Ji Hyoung

Hongik University, Schools of Games, Game Software Major

## 요 약

본 보고서는 Cifar-10 학습 모델과 Python dash 를 이용해 코드를 Refactoring 하였다. Python dash 어플리케이션 내에서 이미지를 업로드하면, 해당 이미지의 예측값 및 각 클래스 레이블 별로 정확도를 텍스트와 그래프로 나타나게 하였다.

### 1)

```
# 모델 정의
2개의 사용 위치
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

[그림 1 - 신경망 모델 정의]

먼저, 컨볼루션 신경망 모델을 정의해준다. _Init_ 메서드에서 신경망의 계층을 초기화하고, 'forward' 메서드에서 입력 데이터의 순방향 전파를 정의한다. 주의해야할 점은, 이미지 분류 클래스 수에 따라 텐서의 크기도 조정을 해주어야 한다.

### 2)

```
def preprocess_image(image):
    transform = transforms.Compose([
        transforms.Resize((32, 32)),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])
    return transform(image).unsqueeze(0)
```

[그림 2 - 이미지 전처리 함수 정의]

CIFAR-10 데이터셋에 적용할 이미지 전처리 함수를 정의해준다. 데이터를 텐서로 변환하고, 정규화시키는 작업을 통해 신경망 모델에 입력할 수 있는 형태로 데이터를 가공시키는 작업이다.

### 3)

```
# 학습된 모델 불러오기
PATH = "model.pth"  # 모델의 경로
net.load_state_dict(torch.load(PATH))
net.eval()
```

[그림 3 - 학습 모델 불러오기]

1) 번에서 정의했던 'Ne't 클래스의 인스턴스를 생성해준 뒤, 사용 가능한 GPU 가 있다면 GPU 로 실행하게 만든다. 또한 학습된 불러와서 모델의 가중치를 로드해준다.

## 4)

```
# 클래스 레이블 정의
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

[그림 4 - 클래스 레이블 정의]

클래스 레이블을 정의해준다. 각 클래스들은 CIFAR-10 데이터셋에 있는 클래스 레이블들과 동일하며, 총 10 개로 이루어져있다.

## 5)

```
# Dash 앱 생성
app = dash.Dash(__name__)

# 이미지 업로드 및 결과 출력 레이아웃
app.layout = html.Div([

    html.H1(children='이미지 업로드를 통한 이미지 분류'),

    dcc.Upload(
        id='upload-image',
        children=html.Div([
            '드래그 앤 드랍하거나 ',
            html.A('이미지 파일을 업로드 해주세요.')
        ]),
        style={
            'width': '50%',
            'height': '50px',
            'lineHeight': '50px',
            'borderWidth': '1px',
            'borderStyle': 'dashed',
            'borderRadius': '5px',
            'textAlign': 'center',
            'margin': '10px'
        },
        multiple=False
    ),
    html.Div(id='output-image-upload'),
    html.Div(id='output-accuracy')
])
```

**[그림 5 - dash 앱 생성 및 레이아웃 정의]**

'dash.Dash'를 사용하여 Dash 애플리케이션을 생성해준다. 업로드 및 결과를 출력하기 위한 레이아웃을 출력하기 위해 dcc.Upload 컴포넌트를 생성해준뒤, style 을 조정해주고, 이미지와 정확도를 출력할 html.div 컴포넌트 또한 생성해준다.

## 6)

```
@app.callback(
    [Output('output-image-upload', 'children'), Output('output-accuracy', 'children')],
    [Input('upload-image', 'contents')],
    [State('upload-image', 'filename')]
)
```

[그림 6 - 이미지 업로드 콜백 함수]

이미지가 입력될 때마다 그 출력값 또한 계속 변하게 만들어주어야 하므로 @app.callback 데코레이터를 사용하여 콜백 함수를 정의해준다. 콜백 함수에 매핑해줄 함수는 무조건 데코레이터 바로 아래에 빈줄 없이 함수를 정의해주어야 한다

## 7)

```
def update_output(contents, filename):
    if contents is not None:
        # base64 인코딩된 이미지 데이터를 디코딩하여 PIL 이미지로 변환
        content_type, content_string = contents.split(',')
        decoded = base64.b64decode(content_string)
        image = Image.open(io.BytesIO(decoded))

        # 이미지 전처리
        processed_image = preprocess_image(image)
        processed_image = processed_image.to(device)

        # 예측
        outputs = net(processed_image)
        _, predicted = torch.max(outputs, 1)
        prediction = classes[predicted.item()]

        # 예측 결과 출력
        result_div = html.Div([
            html.H3('업로드된 이미지:'),
            html.Img(src=contents, style={'height': '300px'}),
            html.H3('Prediction: {}'.format(prediction))
        ])
```

[그림 7 - 매핑할 콜백함수 정의]

데코레이터 아래에 매핑해줄 함수를 정의해준다. 업로드를 해서 contents 인수를 가지고 디코딩하여 이미지로 변환해준 뒤, 이전에 정의해두었던 전처리 함수를 통해 이미지 전처리를 거치고 이미지 예측을 실행한다

## 8)

```
    # 클래스별 정확도 계산
    probabilities = F.softmax(outputs, dim=1)[0].cpu().detach().numpy()
    accuracy_div = html.Div([
        html.H3('각 클래스 별 정확도 :')
    ])
    for i, class_name in enumerate(classes):
        accuracy = probabilities[i]
        accuracy_text = f'{class_name}: {accuracy:.4f}'
        accuracy_div.children.append(html.P(accuracy_text))

    # 클래스별 정확도 그래프
    data = {
        'Class': classes,
        'Accuracy': probabilities
    }
    fig = px.bar(data, x='Class', y='Accuracy')
    graph_div = dcc.Graph(figure=fig)

    return result_div, [accuracy_div, graph_div]
else:
    return None, None
```

[그림 7.1 - 매핑할 콜백함수 정의 2]

예측을 실행하고 나서, 각 클래스별 정확도와 데이터 시각화를 위한 그래프를 출력해주기 위해 여러 컴포넌트를 추가해주고 데코레이터에 수에 맞게 output 을 반환해준다.

## 9)

```
if __name__ == '__main__':
    app.run_server(debug=True)
```

[그림 8 - 앱 실행]

마지막으로, if __name__ == '__main__' 구문을 사용하여 어플리케이션을 실행해준다.