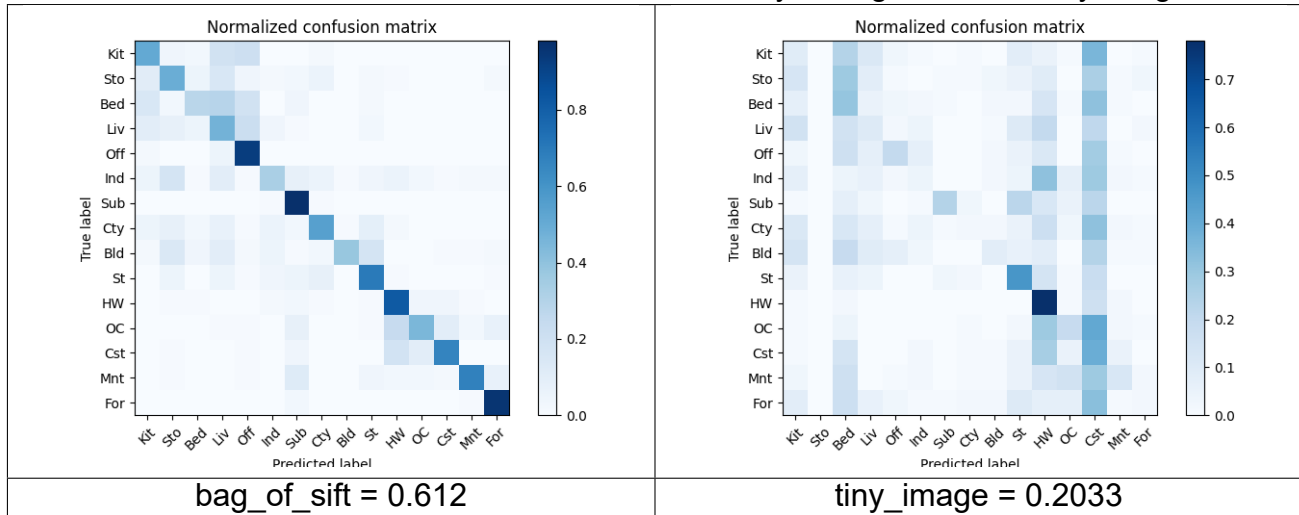


1 Part 1. (10%)

1.1 Plot confusion matrix of two settings. (i.e. Bag of sift and tiny image) (5%)

Table 1: The confusion matrix and the accuracy of bag of sift and tiny image



1.2 Compare the results/accuracy of both settings and explain the result. (5%)

In the course of model training, there's a multitude of parameters ripe for fine-tuning to maximize performance. As a result, models trained with varying parameter setups showcase discrepancies in accuracy and methodological strategies. Presented below are the outcomes observed across different parameter configurations.

1.2.1 tiny_image

Based on the TODO recommendations, I resized the images to $16 * 16$, normalized them, flattened them into one-dimensional vectors, and then calculated the accuracy using the k-nearest neighbors (KNN) algorithm. Within kNN, I utilized `cdist` to compute distances, and subsequently assigned labels to the nearest k neighbors, employing the mode to determine the predicted value. Based on supplementary recommendations, by considering different metrics to evaluate the distances between features, replacing default `metric='euclidean'` with `metric='cityblock'`, the accuracy has shown a noticeable improvement.

However, despite these optimizations, the model still fails to surpass the baseline (0.2). Upon delving deeper into the code and closer inspection, I found that during the normalization process, there is a choice between utilizing either the L1 norm or the L2 norm, with the latter being the default in `np.linalg.norm()` function. As a result, by switching to the L1 norm for normalization, the model successfully exceeded the baseline threshold of 0.2. Furthermore, recognizing the significance of selecting the appropriate value for k in the k-nearest neighbors algorithm, we have conducted experiments using different values of k in search of the optimal solution.

$$\text{euclidean} \Rightarrow d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

$$\text{cityblock} \Rightarrow d(x, y) = |x_1 - y_1| + |x_2 - y_2|$$

$$\text{1-norm} \Rightarrow \|A\|_S = \sum_{i,j} \text{abs}(a_{i,j})$$

$$\text{2-norm} \Rightarrow \|A\|_F = \left[\sum_{i,j} \text{abs}(a_{i,j})^2 \right]^{(1/2)}$$

Table 2: The accuracy of various values of k in the tiny image KNN algorithm

	k=1	k=2	k=3	k=4	k=5
Accuracy	0.2233	0.1787	0.1887	0.2047	0.2020
	k=6	k=7	k=8	k=9	k=10
Accuracy	0.1980	0.2000	0.2033	0.1980	0.2020

1.2.2 bag_of_sift

According to the task description, we understand that our initial step involves extracting feature descriptors from images using dsift, followed by clustering these features to generate a vocabulary (vocab.pkl). Since an image may contain numerous features, we can represent the bag of sift by creating histograms that illustrate the distribution of cluster assignments for features and ultimately classify using KNN.

For the sake of consistency, we keep the same metrics used from tiny images to evaluate the distances, which means only the step size of dsift() function and the step size of kmean() function can be tuned for optimizing the model. When choosing dsift() function parameters, prioritizing a higher number of features per image can enhance the amount of information provided. This typically results in increased accuracy, albeit at the expense of longer computation times.

Consequently, I conducted tests with various sizes, ranging from step=[5, 5], step=[3, 3], down to step=[1, 1], with results aligning as anticipated. Additionally, larger cluster sizes (vocab_size) tend to yield superior performance, hence I retained the default value (400) here. Nonetheless, opting for a cluster size that is too small noticeably impairs the ability to distinguish features effectively. Lastly, the choice of k value was determined through experimentation to achieve optimal results.

Table 3: The accuracy of various values of k in bag of sift KNN algorithm

	k=1	k=2	k=3	k=4	k=5
Accuracy	0.5933	0.5780	0.5820	0.5920	0.5927
	k=6	k=7	k=8	k=9	k=10
Accuracy	0.5947	0.6013	0.6120	0.6033	0.5973

Table 4: The accuracy of various step sizes of dsift function

	step=[1, 1]	step=[3, 3]	step=[5, 5]
k=8	0.6120	0.5833	0.5453

1.2.3 summary

The confusion matrix reveals that the bag of sift approach exhibits a more symmetric matrix compared to tiny images, with only a handful of errors concentrated in the upper-left quadrant. Conversely, in the case of tiny images, a significant proportion of instances are directly classified into specific classes such as Cst, HW, Bed, and Kit, resulting in an asymmetric matrix. Consequently, this observation suggests that the bag of sift approach demonstrates higher accuracy when compared to tiny images.

2 Part 2. (25%)

2.1 Report accuracy of both models on the validation set. (2%)

Table 5: The accuracy and loss of both models on training and validation set

model	Train Acc	Val Acc	Train Loss	Val Loss	Accuracy =
myNet	0.97840	0.88980	0.06605	0.39247	0.8898
resnet18	0.98415	0.89860	0.05136	0.40423	0.8918

2.2 Print the network architecture & number of parameters of both models. What is the main difference between ResNet and other CNN architectures? (5%)

2.2.1 network architecture of mynet model

```

1  MyNet(
2      (conv): Sequential(
3          (0): Conv2d(3, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
4          (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
5          (2): ReLU(inplace=True)
6      )
7      (inception_3a): Inception(
8          (branch1): BasicConv2d(
9              (conv): Conv2d(192, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
10             (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
11         )
12         (branch2): Sequential(
13             (0): BasicConv2d(
14                 (conv): Conv2d(192, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
15                 (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
16             )
17             (1): BasicConv2d(
18                 (conv): Conv2d(96, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
19                 (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
20             )
21         )
22         (branch3): Sequential(
23             (0): BasicConv2d(
24                 (conv): Conv2d(192, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)

```

```

25         (bn): BatchNorm2d(16, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
26     )
27     (1): BasicConv2d(
28         (conv): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
29         (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
30     )
31     (2): BasicConv2d(
32         (conv): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
33         (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
34     )
35 )
36 (branch4): Sequential(
37     (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=
False)
38     (1): BasicConv2d(
39         (conv): Conv2d(192, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
40         (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
41     )
42 )
43 )
44 (inception_3b): Inception(
45     (branch1): BasicConv2d(
46         (conv): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
47         (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
48     )
49     (branch2): Sequential(
50         (0): BasicConv2d(
51             (conv): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
52             (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
53         )
54         (1): BasicConv2d(
55             (conv): Conv2d(128, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
56             (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
57         )
58     )
59     (branch3): Sequential(
60         (0): BasicConv2d(
61             (conv): Conv2d(256, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
62             (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
63         )
64         (1): BasicConv2d(
65             (conv): Conv2d(32, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
66             (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
67         )
68         (2): BasicConv2d(
69             (conv): Conv2d(96, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
70             (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)

```

```

71         )
72     )
73     (branch4): Sequential(
74         (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=
False)
75         (1): BasicConv2d(
76             (conv): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
77             (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
78         )
79     )
80 )
81 (maxpool_3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
82 (inception_4a): Inception(
83     (branch1): BasicConv2d(
84         (conv): Conv2d(480, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
85         (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
86     )
87     (branch2): Sequential(
88         (0): BasicConv2d(
89             (conv): Conv2d(480, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
90             (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
91         )
92         (1): BasicConv2d(
93             (conv): Conv2d(96, 208, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
, bias=False)
94             (bn): BatchNorm2d(208, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
95         )
96     )
97     (branch3): Sequential(
98         (0): BasicConv2d(
99             (conv): Conv2d(480, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
100             (bn): BatchNorm2d(16, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
101         )
102         (1): BasicConv2d(
103             (conv): Conv2d(16, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
104             (bn): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
105         )
106         (2): BasicConv2d(
107             (conv): Conv2d(48, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
108             (bn): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
109         )
110     )
111     (branch4): Sequential(
112         (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=
False)
113         (1): BasicConv2d(
114             (conv): Conv2d(480, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
115             (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
116         )
117     )
118 )

```

```

119         (inception_4b): Inception(
120             (branch1): BasicConv2d(
121                 (conv): Conv2d(512, 160, kernel_size=(1, 1), stride=(1, 1), bias=False)
122                 (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
123             )
124             (branch2): Sequential(
125                 (0): BasicConv2d(
126                     (conv): Conv2d(512, 112, kernel_size=(1, 1), stride=(1, 1), bias=False)
127                     (bn): BatchNorm2d(112, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
128                 )
129                 (1): BasicConv2d(
130                     (conv): Conv2d(112, 224, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1, bias=False)
131                     (bn): BatchNorm2d(224, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
132                 )
133             )
134             (branch3): Sequential(
135                 (0): BasicConv2d(
136                     (conv): Conv2d(512, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
137                     (bn): BatchNorm2d(24, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
138                 )
139                 (1): BasicConv2d(
140                     (conv): Conv2d(24, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
141                     (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
142                 )
143                 (2): BasicConv2d(
144                     (conv): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
145                     (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
146                 )
147             )
148             (branch4): Sequential(
149                 (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=
False)
150                 (1): BasicConv2d(
151                     (conv): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
152                     (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
153                 )
154             )
155         )
156         (inception_4c): Inception(
157             (branch1): BasicConv2d(
158                 (conv): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
159                 (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
160             )
161             (branch2): Sequential(
162                 (0): BasicConv2d(
163                     (conv): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
164                     (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
165                 )
166                 (1): BasicConv2d(

```

```
167         (conv): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
168         1), bias=False)
169         (bn): BatchNorm2d(256, eps=0.001, momentum=0.1, affine=True,
170         track_running_stats=True)
171     )
172     (branch3): Sequential(
173     (0): BasicConv2d(
174     (conv): Conv2d(512, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
175     (bn): BatchNorm2d(24, eps=0.001, momentum=0.1, affine=True,
176     track_running_stats=True)
177     )
178     (1): BasicConv2d(
179     (conv): Conv2d(24, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
180     bias=False)
181     (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
182     track_running_stats=True)
183     )
184     (2): BasicConv2d(
185     (conv): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
186     bias=False)
187     (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
188     track_running_stats=True)
189     )
190     )
191     (branch4): Sequential(
192     (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=
193     False)
194     (1): BasicConv2d(
195     (conv): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
196     (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
197     track_running_stats=True)
198     )
199     )
200     (inception_4d): Inception(
201     (branch1): BasicConv2d(
202     (conv): Conv2d(512, 112, kernel_size=(1, 1), stride=(1, 1), bias=False)
203     (bn): BatchNorm2d(112, eps=0.001, momentum=0.1, affine=True,
204     track_running_stats=True)
205     )
206     (branch2): Sequential(
207     (0): BasicConv2d(
208     (conv): Conv2d(512, 144, kernel_size=(1, 1), stride=(1, 1), bias=False)
209     (bn): BatchNorm2d(144, eps=0.001, momentum=0.1, affine=True,
210     track_running_stats=True)
211     )
212     (1): BasicConv2d(
213     (conv): Conv2d(144, 288, kernel_size=(3, 3), stride=(1, 1), padding=(1,
214     1), bias=False)
215     (bn): BatchNorm2d(288, eps=0.001, momentum=0.1, affine=True,
216     track_running_stats=True)
217     )
218     (branch3): Sequential(
219     (0): BasicConv2d(
220     (conv): Conv2d(512, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
221     (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
222     track_running_stats=True)
223     )
224     (1): BasicConv2d(
```

```

214         (conv): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
    bias=False)
215         (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
216     )
217     (2): BasicConv2d(
218         (conv): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
    bias=False)
219         (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
220     )
221 )
222 (branch4): Sequential(
223     (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=
False)
224     (1): BasicConv2d(
225         (conv): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
226         (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
227     )
228 )
229 )
230 (inception_4e): Inception(
231     (branch1): BasicConv2d(
232         (conv): Conv2d(528, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
233         (bn): BatchNorm2d(256, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
234     )
235     (branch2): Sequential(
236         (0): BasicConv2d(
237             (conv): Conv2d(528, 160, kernel_size=(1, 1), stride=(1, 1), bias=False)
238             (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
239         )
240         (1): BasicConv2d(
241             (conv): Conv2d(160, 320, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
242             (bn): BatchNorm2d(320, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
243         )
244     )
245     (branch3): Sequential(
246         (0): BasicConv2d(
247             (conv): Conv2d(528, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
248             (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
249         )
250         (1): BasicConv2d(
251             (conv): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
    bias=False)
252             (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
253         )
254         (2): BasicConv2d(
255             (conv): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
256             (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
257         )
258     )
259     (branch4): Sequential(

```



```

260         (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=
False)
261         (1): BasicConv2d(
262             (conv): Conv2d(528, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
263             (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
264         )
265     )
266 )
267     (maxpool_4): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
268     (inception_5a): Inception(
269         (branch1): BasicConv2d(
270             (conv): Conv2d(832, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
271             (bn): BatchNorm2d(256, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
272         )
273         (branch2): Sequential(
274             (0): BasicConv2d(
275                 (conv): Conv2d(832, 160, kernel_size=(1, 1), stride=(1, 1), bias=False)
276                 (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
277             )
278             (1): BasicConv2d(
279                 (conv): Conv2d(160, 320, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
280                 (bn): BatchNorm2d(320, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
281             )
282         )
283         (branch3): Sequential(
284             (0): BasicConv2d(
285                 (conv): Conv2d(832, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
286                 (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
287             )
288             (1): BasicConv2d(
289                 (conv): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
, bias=False)
290                 (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
291             )
292             (2): BasicConv2d(
293                 (conv): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
294                 (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
295             )
296         )
297         (branch4): Sequential(
298             (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=
False)
299             (1): BasicConv2d(
300                 (conv): Conv2d(832, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
301                 (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
302             )
303         )
304     )
305     (inception_5b): Inception(
306         (branch1): BasicConv2d(
307             (conv): Conv2d(832, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)

```

```

308         (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
309     )
310     (branch2): Sequential(
311         (0): BasicConv2d(
312             (conv): Conv2d(832, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
313             (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
314         )
315         (1): BasicConv2d(
316             (conv): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
317             (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
318         )
319     )
320     (branch3): Sequential(
321         (0): BasicConv2d(
322             (conv): Conv2d(832, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
323             (bn): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
324         )
325         (1): BasicConv2d(
326             (conv): Conv2d(48, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
, bias=False)
327             (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
328         )
329         (2): BasicConv2d(
330             (conv): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
331             (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
332         )
333     )
334     (branch4): Sequential(
335         (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=
False)
336         (1): BasicConv2d(
337             (conv): Conv2d(832, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
338             (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
339         )
340     )
341 )
342 (avgpool): AvgPool2d(kernel_size=8, stride=1, padding=0)
343 (dropout): Dropout(p=0.2, inplace=False)
344 (fc): Linear(in_features=1024, out_features=10, bias=True)
345 )

```

2.2.2 number of parameters of mynet model

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 192, 32, 32]	5,376
BatchNorm2d-2	[-1, 192, 32, 32]	384
ReLU-3	[-1, 192, 32, 32]	0
Conv2d-4	[-1, 64, 32, 32]	12,288
BatchNorm2d-5	[-1, 64, 32, 32]	128

9	BasicConv2d-6	[-1, 64, 32, 32]	0
10	Conv2d-7	[-1, 96, 32, 32]	18,432
11	BatchNorm2d-8	[-1, 96, 32, 32]	192
12	BasicConv2d-9	[-1, 96, 32, 32]	0
13	Conv2d-10	[-1, 128, 32, 32]	110,592
14	BatchNorm2d-11	[-1, 128, 32, 32]	256
15	BasicConv2d-12	[-1, 128, 32, 32]	0
16	Conv2d-13	[-1, 16, 32, 32]	3,072
17	BatchNorm2d-14	[-1, 16, 32, 32]	32
18	BasicConv2d-15	[-1, 16, 32, 32]	0
19	Conv2d-16	[-1, 32, 32, 32]	4,608
20	BatchNorm2d-17	[-1, 32, 32, 32]	64
21	BasicConv2d-18	[-1, 32, 32, 32]	0
22	Conv2d-19	[-1, 32, 32, 32]	9,216
23	BatchNorm2d-20	[-1, 32, 32, 32]	64
24	BasicConv2d-21	[-1, 32, 32, 32]	0
25	MaxPool2d-22	[-1, 192, 32, 32]	0
26	Conv2d-23	[-1, 32, 32, 32]	6,144
27	BatchNorm2d-24	[-1, 32, 32, 32]	64
28	BasicConv2d-25	[-1, 32, 32, 32]	0
29	Inception-26	[-1, 256, 32, 32]	0
30	Conv2d-27	[-1, 128, 32, 32]	32,768
31	BatchNorm2d-28	[-1, 128, 32, 32]	256
32	BasicConv2d-29	[-1, 128, 32, 32]	0
33	Conv2d-30	[-1, 128, 32, 32]	32,768
34	BatchNorm2d-31	[-1, 128, 32, 32]	256
35	BasicConv2d-32	[-1, 128, 32, 32]	0
36	Conv2d-33	[-1, 192, 32, 32]	221,184
37	BatchNorm2d-34	[-1, 192, 32, 32]	384
38	BasicConv2d-35	[-1, 192, 32, 32]	0
39	Conv2d-36	[-1, 32, 32, 32]	8,192
40	BatchNorm2d-37	[-1, 32, 32, 32]	64
41	BasicConv2d-38	[-1, 32, 32, 32]	0
42	Conv2d-39	[-1, 96, 32, 32]	27,648
43	BatchNorm2d-40	[-1, 96, 32, 32]	192
44	BasicConv2d-41	[-1, 96, 32, 32]	0
45	Conv2d-42	[-1, 96, 32, 32]	82,944
46	BatchNorm2d-43	[-1, 96, 32, 32]	192
47	BasicConv2d-44	[-1, 96, 32, 32]	0
48	MaxPool2d-45	[-1, 256, 32, 32]	0
49	Conv2d-46	[-1, 64, 32, 32]	16,384
50	BatchNorm2d-47	[-1, 64, 32, 32]	128
51	BasicConv2d-48	[-1, 64, 32, 32]	0
52	Inception-49	[-1, 480, 32, 32]	0
53	MaxPool2d-50	[-1, 480, 16, 16]	0
54	Conv2d-51	[-1, 192, 16, 16]	92,160
55	BatchNorm2d-52	[-1, 192, 16, 16]	384
56	BasicConv2d-53	[-1, 192, 16, 16]	0
57	Conv2d-54	[-1, 96, 16, 16]	46,080
58	BatchNorm2d-55	[-1, 96, 16, 16]	192
59	BasicConv2d-56	[-1, 96, 16, 16]	0
60	Conv2d-57	[-1, 208, 16, 16]	179,712
61	BatchNorm2d-58	[-1, 208, 16, 16]	416
62	BasicConv2d-59	[-1, 208, 16, 16]	0
63	Conv2d-60	[-1, 16, 16, 16]	7,680
64	BatchNorm2d-61	[-1, 16, 16, 16]	32
65	BasicConv2d-62	[-1, 16, 16, 16]	0
66	Conv2d-63	[-1, 48, 16, 16]	6,912
67	BatchNorm2d-64	[-1, 48, 16, 16]	96
68	BasicConv2d-65	[-1, 48, 16, 16]	0
69	Conv2d-66	[-1, 48, 16, 16]	20,736
70	BatchNorm2d-67	[-1, 48, 16, 16]	96

71	BasicConv2d-68	[-1, 48, 16, 16]	0
72	MaxPool2d-69	[-1, 480, 16, 16]	0
73	Conv2d-70	[-1, 64, 16, 16]	30,720
74	BatchNorm2d-71	[-1, 64, 16, 16]	128
75	BasicConv2d-72	[-1, 64, 16, 16]	0
76	Inception-73	[-1, 512, 16, 16]	0
77	Conv2d-74	[-1, 160, 16, 16]	81,920
78	BatchNorm2d-75	[-1, 160, 16, 16]	320
79	BasicConv2d-76	[-1, 160, 16, 16]	0
80	Conv2d-77	[-1, 112, 16, 16]	57,344
81	BatchNorm2d-78	[-1, 112, 16, 16]	224
82	BasicConv2d-79	[-1, 112, 16, 16]	0
83	Conv2d-80	[-1, 224, 16, 16]	225,792
84	BatchNorm2d-81	[-1, 224, 16, 16]	448
85	BasicConv2d-82	[-1, 224, 16, 16]	0
86	Conv2d-83	[-1, 24, 16, 16]	12,288
87	BatchNorm2d-84	[-1, 24, 16, 16]	48
88	BasicConv2d-85	[-1, 24, 16, 16]	0
89	Conv2d-86	[-1, 64, 16, 16]	13,824
90	BatchNorm2d-87	[-1, 64, 16, 16]	128
91	BasicConv2d-88	[-1, 64, 16, 16]	0
92	Conv2d-89	[-1, 64, 16, 16]	36,864
93	BatchNorm2d-90	[-1, 64, 16, 16]	128
94	BasicConv2d-91	[-1, 64, 16, 16]	0
95	MaxPool2d-92	[-1, 512, 16, 16]	0
96	Conv2d-93	[-1, 64, 16, 16]	32,768
97	BatchNorm2d-94	[-1, 64, 16, 16]	128
98	BasicConv2d-95	[-1, 64, 16, 16]	0
99	Inception-96	[-1, 512, 16, 16]	0
100	Conv2d-97	[-1, 128, 16, 16]	65,536
101	BatchNorm2d-98	[-1, 128, 16, 16]	256
102	BasicConv2d-99	[-1, 128, 16, 16]	0
103	Conv2d-100	[-1, 128, 16, 16]	65,536
104	BatchNorm2d-101	[-1, 128, 16, 16]	256
105	BasicConv2d-102	[-1, 128, 16, 16]	0
106	Conv2d-103	[-1, 256, 16, 16]	294,912
107	BatchNorm2d-104	[-1, 256, 16, 16]	512
108	BasicConv2d-105	[-1, 256, 16, 16]	0
109	Conv2d-106	[-1, 24, 16, 16]	12,288
110	BatchNorm2d-107	[-1, 24, 16, 16]	48
111	BasicConv2d-108	[-1, 24, 16, 16]	0
112	Conv2d-109	[-1, 64, 16, 16]	13,824
113	BatchNorm2d-110	[-1, 64, 16, 16]	128
114	BasicConv2d-111	[-1, 64, 16, 16]	0
115	Conv2d-112	[-1, 64, 16, 16]	36,864
116	BatchNorm2d-113	[-1, 64, 16, 16]	128
117	BasicConv2d-114	[-1, 64, 16, 16]	0
118	MaxPool2d-115	[-1, 512, 16, 16]	0
119	Conv2d-116	[-1, 64, 16, 16]	32,768
120	BatchNorm2d-117	[-1, 64, 16, 16]	128
121	BasicConv2d-118	[-1, 64, 16, 16]	0
122	Inception-119	[-1, 512, 16, 16]	0
123	Conv2d-120	[-1, 112, 16, 16]	57,344
124	BatchNorm2d-121	[-1, 112, 16, 16]	224
125	BasicConv2d-122	[-1, 112, 16, 16]	0
126	Conv2d-123	[-1, 144, 16, 16]	73,728
127	BatchNorm2d-124	[-1, 144, 16, 16]	288
128	BasicConv2d-125	[-1, 144, 16, 16]	0
129	Conv2d-126	[-1, 288, 16, 16]	373,248
130	BatchNorm2d-127	[-1, 288, 16, 16]	576
131	BasicConv2d-128	[-1, 288, 16, 16]	0
132	Conv2d-129	[-1, 32, 16, 16]	16,384

133	BatchNorm2d-130	[-1, 32, 16, 16]	64
134	BasicConv2d-131	[-1, 32, 16, 16]	0
135	Conv2d-132	[-1, 64, 16, 16]	18,432
136	BatchNorm2d-133	[-1, 64, 16, 16]	128
137	BasicConv2d-134	[-1, 64, 16, 16]	0
138	Conv2d-135	[-1, 64, 16, 16]	36,864
139	BatchNorm2d-136	[-1, 64, 16, 16]	128
140	BasicConv2d-137	[-1, 64, 16, 16]	0
141	MaxPool2d-138	[-1, 512, 16, 16]	0
142	Conv2d-139	[-1, 64, 16, 16]	32,768
143	BatchNorm2d-140	[-1, 64, 16, 16]	128
144	BasicConv2d-141	[-1, 64, 16, 16]	0
145	Inception-142	[-1, 528, 16, 16]	0
146	Conv2d-143	[-1, 256, 16, 16]	135,168
147	BatchNorm2d-144	[-1, 256, 16, 16]	512
148	BasicConv2d-145	[-1, 256, 16, 16]	0
149	Conv2d-146	[-1, 160, 16, 16]	84,480
150	BatchNorm2d-147	[-1, 160, 16, 16]	320
151	BasicConv2d-148	[-1, 160, 16, 16]	0
152	Conv2d-149	[-1, 320, 16, 16]	460,800
153	BatchNorm2d-150	[-1, 320, 16, 16]	640
154	BasicConv2d-151	[-1, 320, 16, 16]	0
155	Conv2d-152	[-1, 32, 16, 16]	16,896
156	BatchNorm2d-153	[-1, 32, 16, 16]	64
157	BasicConv2d-154	[-1, 32, 16, 16]	0
158	Conv2d-155	[-1, 128, 16, 16]	36,864
159	BatchNorm2d-156	[-1, 128, 16, 16]	256
160	BasicConv2d-157	[-1, 128, 16, 16]	0
161	Conv2d-158	[-1, 128, 16, 16]	147,456
162	BatchNorm2d-159	[-1, 128, 16, 16]	256
163	BasicConv2d-160	[-1, 128, 16, 16]	0
164	MaxPool2d-161	[-1, 528, 16, 16]	0
165	Conv2d-162	[-1, 128, 16, 16]	67,584
166	BatchNorm2d-163	[-1, 128, 16, 16]	256
167	BasicConv2d-164	[-1, 128, 16, 16]	0
168	Inception-165	[-1, 832, 16, 16]	0
169	MaxPool2d-166	[-1, 832, 8, 8]	0
170	Conv2d-167	[-1, 256, 8, 8]	212,992
171	BatchNorm2d-168	[-1, 256, 8, 8]	512
172	BasicConv2d-169	[-1, 256, 8, 8]	0
173	Conv2d-170	[-1, 160, 8, 8]	133,120
174	BatchNorm2d-171	[-1, 160, 8, 8]	320
175	BasicConv2d-172	[-1, 160, 8, 8]	0
176	Conv2d-173	[-1, 320, 8, 8]	460,800
177	BatchNorm2d-174	[-1, 320, 8, 8]	640
178	BasicConv2d-175	[-1, 320, 8, 8]	0
179	Conv2d-176	[-1, 32, 8, 8]	26,624
180	BatchNorm2d-177	[-1, 32, 8, 8]	64
181	BasicConv2d-178	[-1, 32, 8, 8]	0
182	Conv2d-179	[-1, 128, 8, 8]	36,864
183	BatchNorm2d-180	[-1, 128, 8, 8]	256
184	BasicConv2d-181	[-1, 128, 8, 8]	0
185	Conv2d-182	[-1, 128, 8, 8]	147,456
186	BatchNorm2d-183	[-1, 128, 8, 8]	256
187	BasicConv2d-184	[-1, 128, 8, 8]	0
188	MaxPool2d-185	[-1, 832, 8, 8]	0
189	Conv2d-186	[-1, 128, 8, 8]	106,496
190	BatchNorm2d-187	[-1, 128, 8, 8]	256
191	BasicConv2d-188	[-1, 128, 8, 8]	0
192	Inception-189	[-1, 832, 8, 8]	0
193	Conv2d-190	[-1, 384, 8, 8]	319,488
194	BatchNorm2d-191	[-1, 384, 8, 8]	768

195	BasicConv2d-192	[-1, 384, 8, 8]	0
196	Conv2d-193	[-1, 192, 8, 8]	159,744
197	BatchNorm2d-194	[-1, 192, 8, 8]	384
198	BasicConv2d-195	[-1, 192, 8, 8]	0
199	Conv2d-196	[-1, 384, 8, 8]	663,552
200	BatchNorm2d-197	[-1, 384, 8, 8]	768
201	BasicConv2d-198	[-1, 384, 8, 8]	0
202	Conv2d-199	[-1, 48, 8, 8]	39,936
203	BatchNorm2d-200	[-1, 48, 8, 8]	96
204	BasicConv2d-201	[-1, 48, 8, 8]	0
205	Conv2d-202	[-1, 128, 8, 8]	55,296
206	BatchNorm2d-203	[-1, 128, 8, 8]	256
207	BasicConv2d-204	[-1, 128, 8, 8]	0
208	Conv2d-205	[-1, 128, 8, 8]	147,456
209	BatchNorm2d-206	[-1, 128, 8, 8]	256
210	BasicConv2d-207	[-1, 128, 8, 8]	0
211	MaxPool2d-208	[-1, 832, 8, 8]	0
212	Conv2d-209	[-1, 128, 8, 8]	106,496
213	BatchNorm2d-210	[-1, 128, 8, 8]	256
214	BasicConv2d-211	[-1, 128, 8, 8]	0
215	Inception-212	[-1, 1024, 8, 8]	0
216	AvgPool2d-213	[-1, 1024, 1, 1]	0
217	Linear-214	[-1, 10]	10,250

```

=====
Total params: 6,158,538
Trainable params: 6,158,538
Non-trainable params: 0
-----

```

```

Input size (MB): 0.01
Forward/backward pass size (MB): 81.42
Params size (MB): 23.49
Estimated Total Size (MB): 104.93
-----

```

2.2.3 network architecture of resnet18 model

```

1  ResNet18(
2      (resnet): ResNet(
3          (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(3, 3),
4              bias=False)
5          (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
6              track_running_stats=True)
7          (relu): ReLU(inplace=True)
8          (maxpool): Identity()
9          (layer1): Sequential(
10             (0): BasicBlock(
11                 (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
12                     bias=False)
13                 (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
14                     track_running_stats=True)
15                 (relu): ReLU(inplace=True)
16                 (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
17                     bias=False)
18                 (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
19                     track_running_stats=True)
20             )
21             (1): BasicBlock(
22                 (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
23                     bias=False)

```

```

17         (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
18         (relu): ReLU(inplace=True)
19         (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
, bias=False)
20         (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
21     )
22 )
23     (layer2): Sequential(
24         (0): BasicBlock(
25             (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
26             (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
27             (relu): ReLU(inplace=True)
28             (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
29             (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
30             (downsample): Sequential(
31                 (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
32                 (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
33             )
34         )
35         (1): BasicBlock(
36             (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
37             (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
38             (relu): ReLU(inplace=True)
39             (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
40             (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
41         )
42     )
43     (layer3): Sequential(
44         (0): BasicBlock(
45             (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
46             (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
47             (relu): ReLU(inplace=True)
48             (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
49             (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
50             (downsample): Sequential(
51                 (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
52                 (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
53             )
54         )
55         (1): BasicBlock(
56             (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
57             (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
58             (relu): ReLU(inplace=True)

```



```

59         (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
60         1), bias=False)
61         (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
62         track_running_stats=True)
63     )
64     (layer4): Sequential(
65         (0): BasicBlock(
66             (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
67             1), bias=False)
68             (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
69             track_running_stats=True)
70             (relu): ReLU(inplace=True)
71             (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
72             1), bias=False)
73             (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
74             track_running_stats=True)
75             (downsample): Sequential(
76                 (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
77                 (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
78                 track_running_stats=True)
79             )
80             (1): BasicBlock(
81                 (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
82                 1), bias=False)
83                 (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
84                 track_running_stats=True)
85                 (relu): ReLU(inplace=True)
86                 (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
87                 1), bias=False)
88                 (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
89                 track_running_stats=True)
90             )
91         )
92     )
93     (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
94     (fc): Linear(in_features=512, out_features=10, bias=True)
95 )

```

2.2.4 number of parameters of resnet18 model

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 36, 36]	1,728
BatchNorm2d-2	[-1, 64, 36, 36]	128
ReLU-3	[-1, 64, 36, 36]	0
Identity-4	[-1, 64, 36, 36]	0
Conv2d-5	[-1, 64, 36, 36]	36,864
BatchNorm2d-6	[-1, 64, 36, 36]	128
ReLU-7	[-1, 64, 36, 36]	0
Conv2d-8	[-1, 64, 36, 36]	36,864
BatchNorm2d-9	[-1, 64, 36, 36]	128
ReLU-10	[-1, 64, 36, 36]	0
BasicBlock-11	[-1, 64, 36, 36]	0
Conv2d-12	[-1, 64, 36, 36]	36,864
BatchNorm2d-13	[-1, 64, 36, 36]	128
ReLU-14	[-1, 64, 36, 36]	0
Conv2d-15	[-1, 64, 36, 36]	36,864

19	BatchNorm2d-16	[-1, 64, 36, 36]	128
20	ReLU-17	[-1, 64, 36, 36]	0
21	BasicBlock-18	[-1, 64, 36, 36]	0
22	Conv2d-19	[-1, 128, 18, 18]	73,728
23	BatchNorm2d-20	[-1, 128, 18, 18]	256
24	ReLU-21	[-1, 128, 18, 18]	0
25	Conv2d-22	[-1, 128, 18, 18]	147,456
26	BatchNorm2d-23	[-1, 128, 18, 18]	256
27	Conv2d-24	[-1, 128, 18, 18]	8,192
28	BatchNorm2d-25	[-1, 128, 18, 18]	256
29	ReLU-26	[-1, 128, 18, 18]	0
30	BasicBlock-27	[-1, 128, 18, 18]	0
31	Conv2d-28	[-1, 128, 18, 18]	147,456
32	BatchNorm2d-29	[-1, 128, 18, 18]	256
33	ReLU-30	[-1, 128, 18, 18]	0
34	Conv2d-31	[-1, 128, 18, 18]	147,456
35	BatchNorm2d-32	[-1, 128, 18, 18]	256
36	ReLU-33	[-1, 128, 18, 18]	0
37	BasicBlock-34	[-1, 128, 18, 18]	0
38	Conv2d-35	[-1, 256, 9, 9]	294,912
39	BatchNorm2d-36	[-1, 256, 9, 9]	512
40	ReLU-37	[-1, 256, 9, 9]	0
41	Conv2d-38	[-1, 256, 9, 9]	589,824
42	BatchNorm2d-39	[-1, 256, 9, 9]	512
43	Conv2d-40	[-1, 256, 9, 9]	32,768
44	BatchNorm2d-41	[-1, 256, 9, 9]	512
45	ReLU-42	[-1, 256, 9, 9]	0
46	BasicBlock-43	[-1, 256, 9, 9]	0
47	Conv2d-44	[-1, 256, 9, 9]	589,824
48	BatchNorm2d-45	[-1, 256, 9, 9]	512
49	ReLU-46	[-1, 256, 9, 9]	0
50	Conv2d-47	[-1, 256, 9, 9]	589,824
51	BatchNorm2d-48	[-1, 256, 9, 9]	512
52	ReLU-49	[-1, 256, 9, 9]	0
53	BasicBlock-50	[-1, 256, 9, 9]	0
54	Conv2d-51	[-1, 512, 5, 5]	1,179,648
55	BatchNorm2d-52	[-1, 512, 5, 5]	1,024
56	ReLU-53	[-1, 512, 5, 5]	0
57	Conv2d-54	[-1, 512, 5, 5]	2,359,296
58	BatchNorm2d-55	[-1, 512, 5, 5]	1,024
59	Conv2d-56	[-1, 512, 5, 5]	131,072
60	BatchNorm2d-57	[-1, 512, 5, 5]	1,024
61	ReLU-58	[-1, 512, 5, 5]	0
62	BasicBlock-59	[-1, 512, 5, 5]	0
63	Conv2d-60	[-1, 512, 5, 5]	2,359,296
64	BatchNorm2d-61	[-1, 512, 5, 5]	1,024
65	ReLU-62	[-1, 512, 5, 5]	0
66	Conv2d-63	[-1, 512, 5, 5]	2,359,296
67	BatchNorm2d-64	[-1, 512, 5, 5]	1,024
68	ReLU-65	[-1, 512, 5, 5]	0
69	BasicBlock-66	[-1, 512, 5, 5]	0
70	AdaptiveAvgPool2d-67	[-1, 512, 1, 1]	0
71	Linear-68	[-1, 10]	5,130
72	ResNet-69	[-1, 10]	0

=====

Total params: 11,173,962

Trainable params: 11,173,962

Non-trainable params: 0

Input size (MB): 0.01

Forward/backward pass size (MB): 20.55

Params size (MB): 42.63

81 Estimated Total Size (MB): 63.19

82 -----

2.2.5 What is the main difference between ResNet and other CNN architectures?

Compared with ResNet, traditional CNN architectures such as LeNet, AlexNet, and VGG, relied on stacking convolutional layers one after another to learn hierarchical features from input images. As the network depth increased, they faced challenges with vanishing gradients and degradation in training performance. ResNet addressed this issue by introducing shortcut connections and the residual learning approach. These connections enable the network to bypass one or more layers, allowing the flow of information directly from shallower layers to deeper layers. This approach alleviates the vanishing gradient problem by providing an alternate, shorter path for gradient flow during training. As a result, ResNet can effectively train much deeper networks by mitigating the vanishing gradient problem, this leads to better performance and accuracy.

2.3 Plot four learning curves (loss & accuracy) of the training process (train/validation) for both models. Total 8 plots. (8%)

Table 6: The learning curves of the training process (train/validation) for mynet models

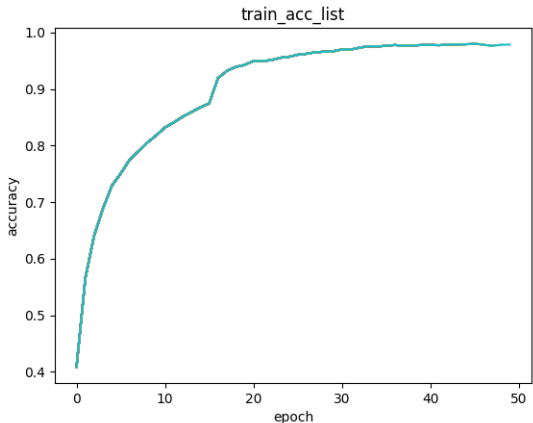
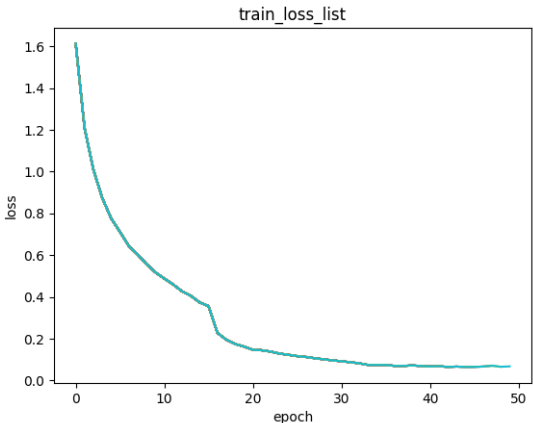
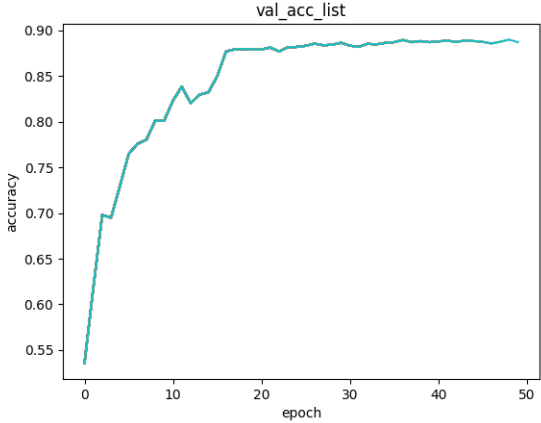
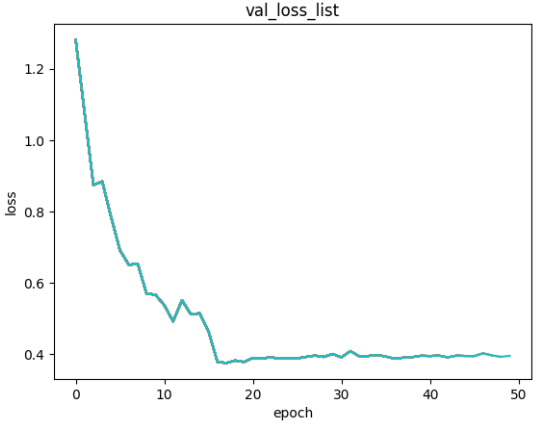
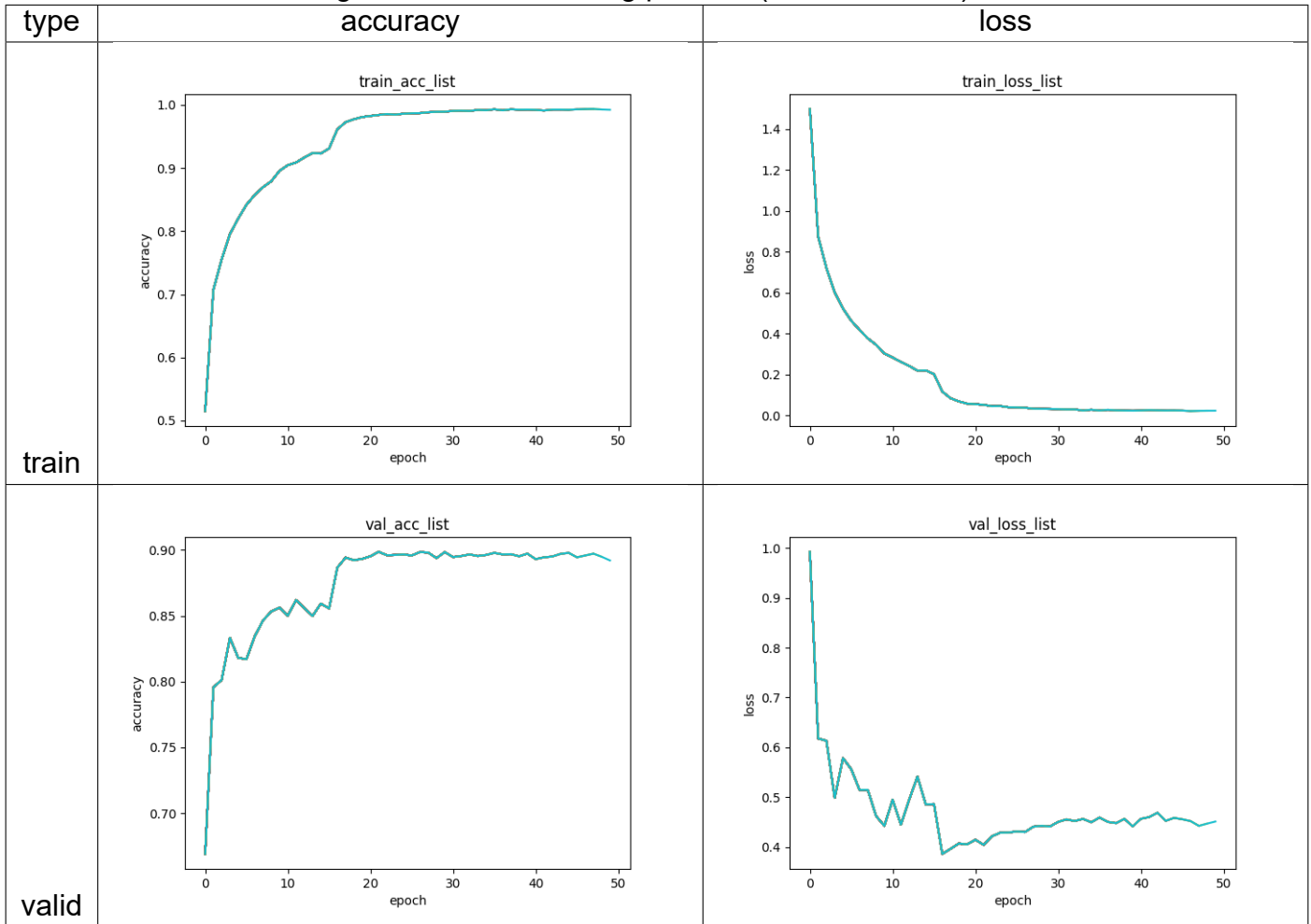
type	accuracy	loss
train		
valid		

Table 7: The learning curves of the training process (train/validation) for resnet18 models



2.4 Briefly describe what method do you apply on your best model? (e.g. data augmentation, model architecture, loss function, etc) (10%)

To ensure the stability of the program, I initially focused on developing and optimizing a ResNet18 model. Considering the rotational invariance challenge of CNNs, I introduced random rotation into the data augmentation process. Following recommendations from the task guidelines, I optimized the model by reducing the kernel size and stride of the first convolution layer, while also eliminating the first max-pooling layer and replacing it with Identity(). Surprisingly, these adjustments resulted in the model passing the strong baseline (0.84) in its first training iteration.

In pursuit of comparable accuracy and performance to the resnet18, I evaluated the VGG16[1] and VGG19[1] CNN models, given their similar release times compared to resnet18. Notably, I observed that VGG19 exhibited lower accuracy compared to VGG16 during the training process. To address the failure to pass the strong baseline, I attempted to optimize the VGG16 model by introducing the dropout strategy and increasing the dimensions of the first two convolutional layers from 64 to 96, this led to an increment of accuracy from 0.825 to 0.8316.

Despite several modifications, optimizing the model architecture alone did not yield the desired accuracy enhancements. Hence, I incorporated data augmentation techniques into the training process again and found that applying random horizontal flip techniques to images consistently improved model accuracy. With this addition, the model surpassed the strong baseline (0.84). However, I noticed that the model size exceeds the 80MB size limitation, so I began exploring all other known CNN models to address this constraint.

The CNN models that I experimented with included LeNet[2], AlexNet[3], GoogleNet[4], and DenseNet[5]. The results revealed that LeNet and AlexNet had relatively low accuracies and failed to meet the strong baseline (0.84). Only GoogleNet and DenseNet surpassed the strong baseline (0.84), achieving accuracies of 0.8898 and 0.876, respectively. Consequently, I opted for GoogleNet as the final model choice.

To improve the GoogleNet model, I attempted to fine-tune its architecture. Firstly, I consolidated the preceding convolution layers into a single convolution layer. Inspired by the advancements in architectures like Inception V2, V3, and V4, one extra layer is added to the convolution block of branch 3. Additionally, I referenced PyTorch to adjust the kernel size from 5 to 3 on branch 3. Finally, I introduced dropout to mitigate superfluous information, ultimately producing my customized version of the GoogleNet model.

References

- [1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.
- [2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.
- [4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.
- [5] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2018.