2024_AI_training_MID_Report

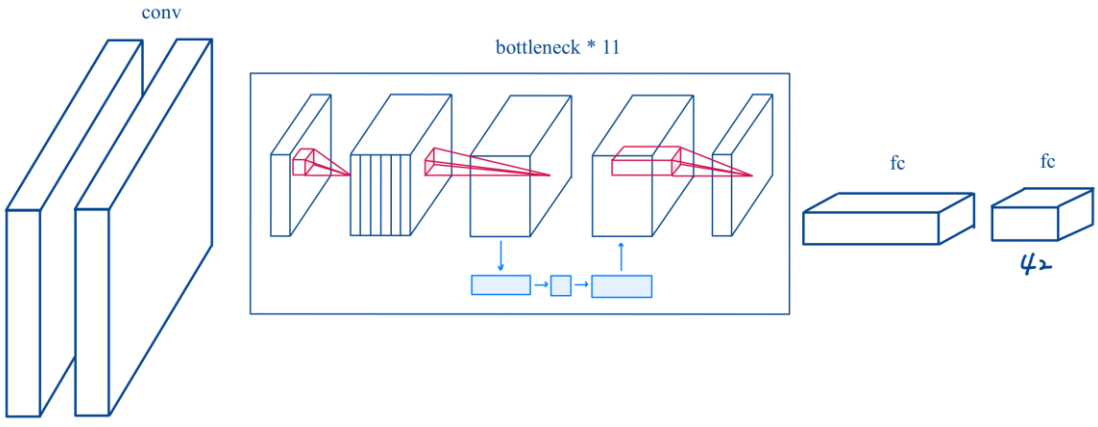
110511118 陳孟頡

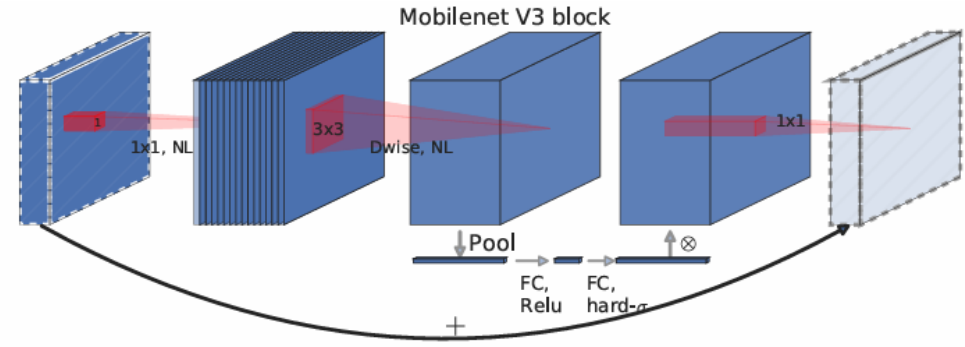
Part I. Model Architecture and Training

- **Model Architecture**

  此次作業採用 MobileNet V3-small，其架構如下:

Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d, 3x3	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	✓	RE	2
$56^2 \times 16$	bneck, 3x3	72	24	-	RE	2
$28^2 \times 24$	bneck, 3x3	88	24	-	RE	1
$28^2 \times 24$	bneck, 5x5	96	40	✓	HS	2
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	120	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	144	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	288	96	✓	HS	2
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	conv2d, 1x1	-	576	✓	HS	1
$7^2 \times 576$	pool, 7x7	-	-	-	-	1
$1^2 \times 576$	conv2d 1x1, NBN	-	1024	-	HS	1
$1^2 \times 1024$	conv2d 1x1, NBN	-	k	-	-	1





以下針對 MobileNet V3 block 中所用到方法進行介紹:

- Depth-wise separable convolution:

  MobileNet 中將傳統 convolution 拆為兩步進行以減少參數量及計算量。首先進行 depth-wise convolution 調整 feature map 大小，接著以 point-wise convolution 調整 dimension 至所需大小。

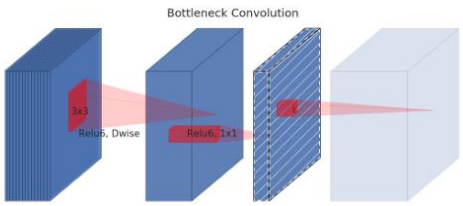
  以此方法，能將原先 input channel=M、output channel=N、kernel size=D_k、feature map size=D_f 的 convolution 計算量從 D_k x D_k x M x N x D_f x D_f 縮減至 (D_k x D_k + N) x M x D_f x D_f

- Linear bottleneck and inverted residual structure:

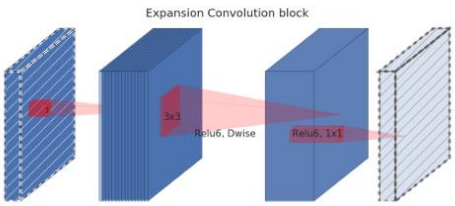
  此為 MobileNet V2 中所提出的架構，先在高維度中擷取特徵資訊，再於低維度中進行傳遞，以更進一步增加模型架構的計算效率。

  Linear bottleneck 與原先 depth-wise separable convolution 不同之處，在於原先的 point-wise convolution 將 channel size 增加，在 linear bottleneck 中則會將 channel size 縮小，把重要資訊 embedding 至較低維度的子空間，來進行傳遞，如下圖(c)。進入下一次的 convolution 前，再由 expansion layer 將 channel size 進行擴張，如下圖(d)，如此一來，便能達到更有效率的資料特徵萃取與傳遞。

(c) Separable with linear bottleneck



(d) Bottleneck with expansion layer



- Squeeze and excitation:

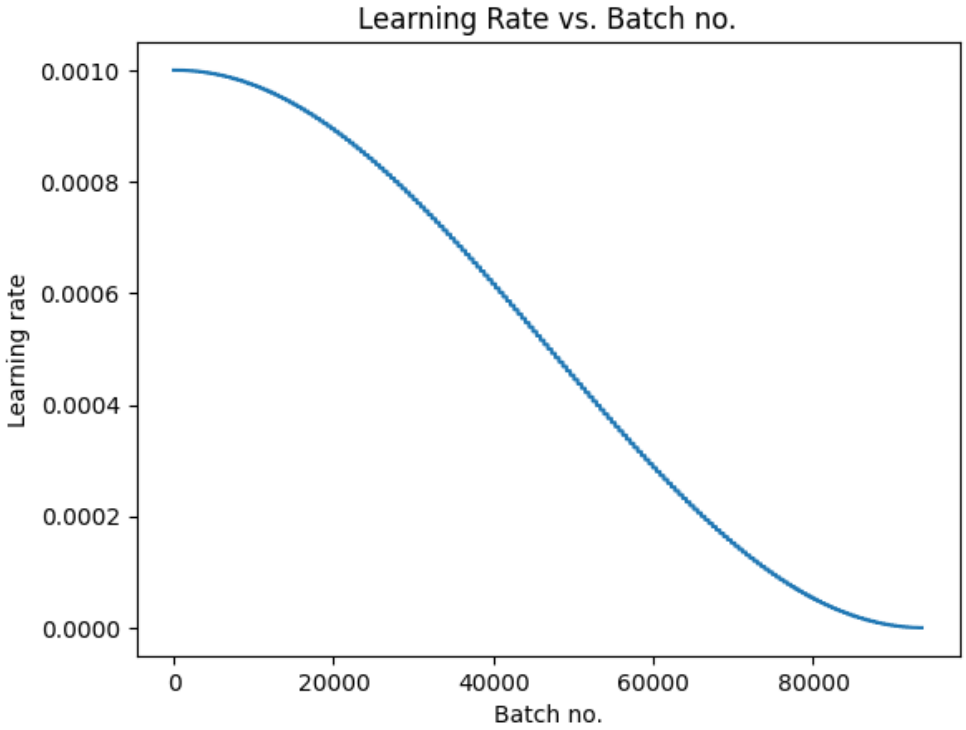
  Squeeze and excitation 目的為在模型架構中加入 attention 機制，進一步提升模型準確度。

  此方法由 squeeze 和 excitation 兩步驟組成。在 squeeze 中，會將原先維度為 H*W*C 的 feature map 壓縮為 1*1*C，讓此一維參數擁有 H*W 的 receptive field。

Excitation 步驟，會將前面獲得的一維參數輸入 fully connected layer，對每個 channel 進行重要性預測，最後再將所得與 squeeze 前的 feature map 進行相乘，完成在 channel 維度的特徵重標定。

- **Training**
  - Optimizer:

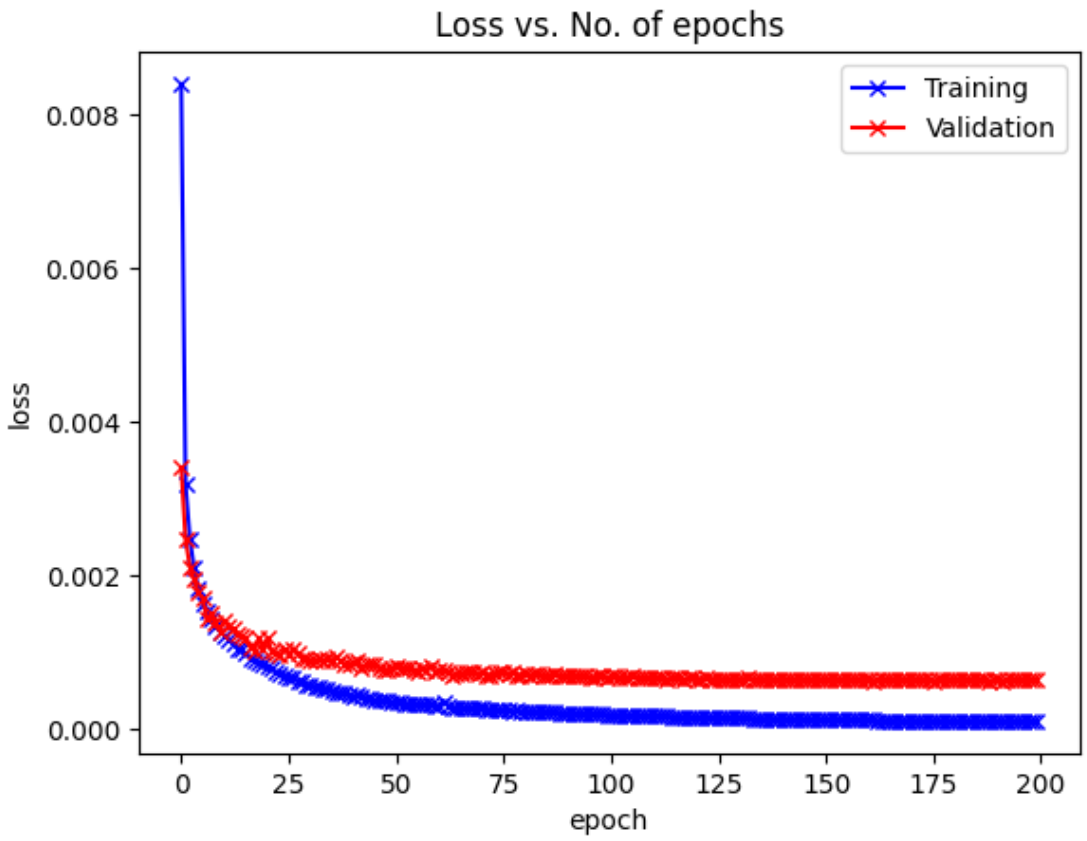
    經過嘗試後，最後採用的 optimizer 為 Adam，scheduler 使用 CosineAnnealingLR，learning rate 變化如下圖:



在嘗試不同的 scheduler 後，發現使用 CosineAnnealingLR 能使模型達到最佳的預測準確度。與其他 scheduler 如 ReduceLROnPlateau 對比，ReduceLROnPlateau 會在 loss 維持一定次數後便會更新，每次變化幅度較大，而 CosineAnnealingLR 則是會隨著時間減少 learning rate，推測在 MobileNet V3 中，連續小幅度的 learning rate 變化，較能使參數更新達到 error surface 的 local minima，ReduceLROnPlateau 的大幅變化反而無法達到對應的表現。

■ Training loss:

Training loss 和 validation loss 變化圖如下:



可以發現 validation loss 在 epoch 數超過 10 後，總是大於 training loss，造成此現象的原因有幾點以下兩點:

1. Overfitting:

   在訓練過程中，模型針對 training dataset 進行最佳化，對於其中資料的特徵較為熟悉，預測能力較強。而 validation dataset 則是沒有看過的資料，因此無法有如 training dataset 的精準度。

2. Regularization:

   在此次所使用的模型中，有加入 dropout 來防止模型過度 overfitting，而 dropout 會使模型在 training 時有較佳的 generalize 能力，。而在 validation 時，dropout 關閉，因此導致 validation loss 大於 training loss。

Part II. Accuracy Improvement

- **Model choosing:**

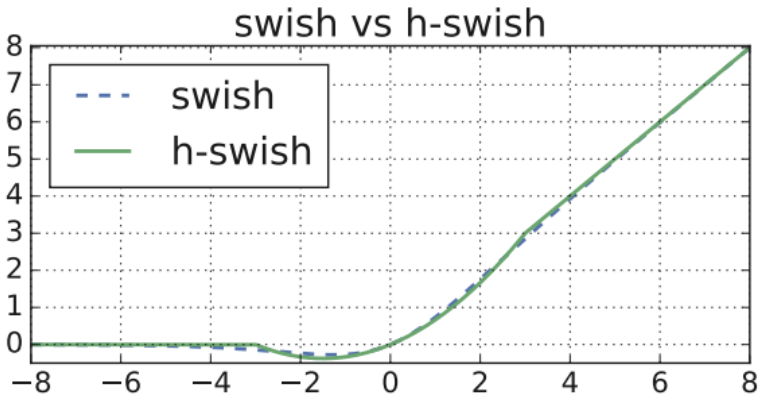
  此次 midterm project 中所選擇的是 MobileNet V3 模型，與其他使用過模型比較如下:

比較模型	模型問題	MobileNet V3 優點
MobileNet V2:	準確度低	加入 Senet 等架構，提升準確度
MobileNet V4:	參數較多，且容易產生嚴重 overfitting	參數少，模型訓練穩定
Transformer-based	參數及計算量皆高	在低參數與計算量下，有一定的準確度
不採用✗		採用✓

- **Activation function:**

  由於 MobileNet V3 中使用到 linear bottleneck and inverted residual structure，此架構透過較小的維度傳遞特徵資訊，若使用 ReLU activation fumction 容易造成重要資訊遺失，因此會將 activation function 換為 hardswish，使用 non-linear function 增加函式複雜程度。

  而使用 hardswish 而不是 swish 的原因在於，hardswish 是 piecewise linearity 版本的 swish，跟 swish 相比減少了計算量，能進一步增加模型運算效率。



- **Data augmentation:**

  將訓練資料進行預處理，降低 overfitting 發生同時提升準確度。所使用 data augmentation 如下:

```
self.preprocess = transforms.Compose([
  transforms.ToPILImage(),
  transforms.Resize(256),
  transforms.CenterCrop(224),
  transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
  transforms.ToTensor(),
  transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

```
image_name = self.image_names[idx]
image = cv2.imread(image_name)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image = self.preprocess(image)
```

- **Optimizer:**

比較項目	RMSprop	Adam
Core Concept	參考以往 learning rate 進行更新	結合 RMSprop 與 momentum，並再更新時考慮 bias-correction
Convergence and Performance	收斂快速，適合處理複雜、non-convex 的 error surface，但容易受到最初 learning rate 影響結果	收斂速度也快，對所有不同的參數都有新舊之間的權衡調節，各種狀況均適用
Hyperparameter Sensitivity	需仔細的調整 learning rate 及 decay rate	較為 robust
Implement Result	Loss 容易大幅度跳動	Loss 穩定下降
	不採用✗	採用✓

- **Learning rate:**

  如前述，利用 CosineAnnealingLR 的 scheduler 來更新 learning rate。

Part III. Project Result

- **Avg PixelDiff**

  6856.40465 / 1000 = **6.85640465**

- **Ranking formula**

  Param: 1.560906 M

  FLOPs: 0.122993904 G

  Avg PixelDiff: 6.85640465

  $1.560906 * 0.122993904 * e^{6.85640465} =$ **182.3724**

- **Model summary**

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 16, 112, 112]	432
BatchNorm2d-2	[-1, 16, 112, 112]	32
Hardswish-3	[-1, 16, 112, 112]	0
Conv2d-4	[-1, 16, 56, 56]	144
BatchNorm2d-5	[-1, 16, 56, 56]	32
ReLU-6	[-1, 16, 56, 56]	0
AdaptiveAvgPool2d-7	[-1, 16, 1, 1]	0
Conv2d-8	[-1, 8, 1, 1]	136
ReLU-9	[-1, 8, 1, 1]	0
Conv2d-10	[-1, 16, 1, 1]	144
Hardsigmoid-11	[-1, 16, 1, 1]	0
SqueezeExcitation-12	[-1, 16, 56, 56]	0
Conv2d-13	[-1, 16, 56, 56]	256
BatchNorm2d-14	[-1, 16, 56, 56]	32
InvertedResidual-15	[-1, 16, 56, 56]	0
Conv2d-16	[-1, 72, 56, 56]	1,152
BatchNorm2d-17	[-1, 72, 56, 56]	144
ReLU-18	[-1, 72, 56, 56]	0
Conv2d-19	[-1, 72, 28, 28]	648
BatchNorm2d-20	[-1, 72, 28, 28]	144
ReLU-21	[-1, 72, 28, 28]	0
Conv2d-22	[-1, 24, 28, 28]	1,728
BatchNorm2d-23	[-1, 24, 28, 28]	48
InvertedResidual-24	[-1, 24, 28, 28]	0
Conv2d-25	[-1, 88, 28, 28]	2,112
BatchNorm2d-26	[-1, 88, 28, 28]	176
ReLU-27	[-1, 88, 28, 28]	0

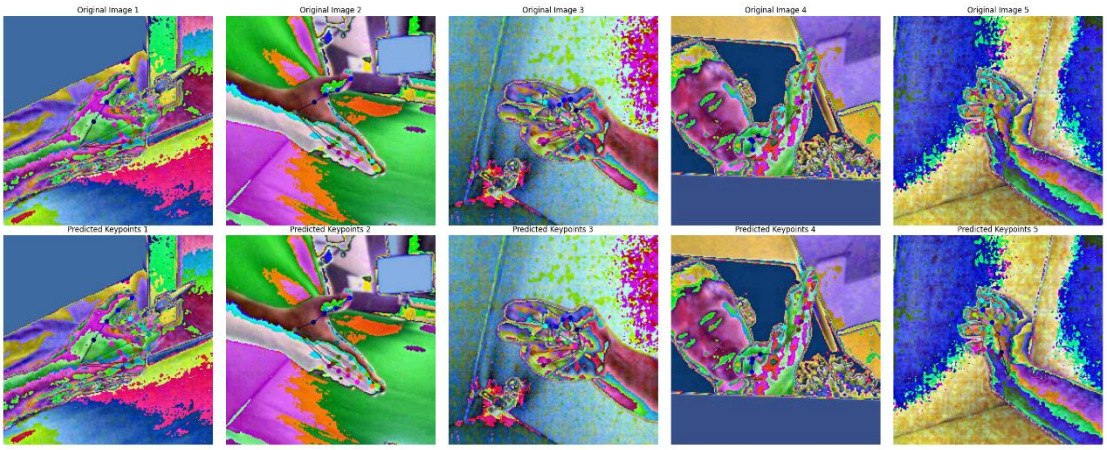
Conv2d-28	[-1, 88, 28, 28]	792
BatchNorm2d-29	[-1, 88, 28, 28]	176
ReLU-30	[-1, 88, 28, 28]	0
Conv2d-31	[-1, 24, 28, 28]	2,112
BatchNorm2d-32	[-1, 24, 28, 28]	48
InvertedResidual-33	[-1, 24, 28, 28]	0
Conv2d-34	[-1, 96, 28, 28]	2,304
BatchNorm2d-35	[-1, 96, 28, 28]	192
Hardswish-36	[-1, 96, 28, 28]	0
Conv2d-37	[-1, 96, 14, 14]	2,400
BatchNorm2d-38	[-1, 96, 14, 14]	192
Hardswish-39	[-1, 96, 14, 14]	0
AdaptiveAvgPool2d-40	[-1, 96, 1, 1]	0
Conv2d-41	[-1, 24, 1, 1]	2,328
ReLU-42	[-1, 24, 1, 1]	0
Conv2d-43	[-1, 96, 1, 1]	2,400
Hardsigmoid-44	[-1, 96, 1, 1]	0
SqueezeExcitation-45	[-1, 96, 14, 14]	0
Conv2d-46	[-1, 40, 14, 14]	3,840
BatchNorm2d-47	[-1, 40, 14, 14]	80
InvertedResidual-48	[-1, 40, 14, 14]	0
Conv2d-49	[-1, 240, 14, 14]	9,600
BatchNorm2d-50	[-1, 240, 14, 14]	480
Hardswish-51	[-1, 240, 14, 14]	0
Conv2d-52	[-1, 240, 14, 14]	6,000
BatchNorm2d-53	[-1, 240, 14, 14]	480
Hardswish-54	[-1, 240, 14, 14]	0
AdaptiveAvgPool2d-55	[-1, 240, 1, 1]	0
Conv2d-56	[-1, 64, 1, 1]	15,424
ReLU-57	[-1, 64, 1, 1]	0
Conv2d-58	[-1, 240, 1, 1]	15,600
Hardsigmoid-59	[-1, 240, 1, 1]	0
SqueezeExcitation-60	[-1, 240, 14, 14]	0
Conv2d-61	[-1, 40, 14, 14]	9,600
BatchNorm2d-62	[-1, 40, 14, 14]	80
InvertedResidual-63	[-1, 40, 14, 14]	0
Conv2d-64	[-1, 240, 14, 14]	9,600
BatchNorm2d-65	[-1, 240, 14, 14]	480
Hardswish-66	[-1, 240, 14, 14]	0
Conv2d-67	[-1, 240, 14, 14]	6,000
BatchNorm2d-68	[-1, 240, 14, 14]	480
Hardswish-69	[-1, 240, 14, 14]	0
AdaptiveAvgPool2d-70	[-1, 240, 1, 1]	0
Conv2d-71	[-1, 64, 1, 1]	15,424
ReLU-72	[-1, 64, 1, 1]	0
Conv2d-73	[-1, 240, 1, 1]	15,600
Hardsigmoid-74	[-1, 240, 1, 1]	0
SqueezeExcitation-75	[-1, 240, 14, 14]	0
Conv2d-76	[-1, 40, 14, 14]	9,600
BatchNorm2d-77	[-1, 40, 14, 14]	80
InvertedResidual-78	[-1, 40, 14, 14]	0
Conv2d-79	[-1, 120, 14, 14]	4,800
BatchNorm2d-80	[-1, 120, 14, 14]	240
Hardswish-81	[-1, 120, 14, 14]	0
Conv2d-82	[-1, 120, 14, 14]	3,000
BatchNorm2d-83	[-1, 120, 14, 14]	240
Hardswish-84	[-1, 120, 14, 14]	0
AdaptiveAvgPool2d-85	[-1, 120, 1, 1]	0
Conv2d-86	[-1, 32, 1, 1]	3,872
ReLU-87	[-1, 32, 1, 1]	0

Conv2d-88	[-1, 120, 1, 1]	3,960
Hardsigmoid-89	[-1, 120, 1, 1]	0
SqueezeExcitation-90	[-1, 120, 14, 14]	0
Conv2d-91	[-1, 48, 14, 14]	5,760
BatchNorm2d-92	[-1, 48, 14, 14]	96
InvertedResidual-93	[-1, 48, 14, 14]	0
Conv2d-94	[-1, 144, 14, 14]	6,912
BatchNorm2d-95	[-1, 144, 14, 14]	288
Hardswish-96	[-1, 144, 14, 14]	0
Conv2d-97	[-1, 144, 14, 14]	3,600
BatchNorm2d-98	[-1, 144, 14, 14]	288
Hardswish-99	[-1, 144, 14, 14]	0
AdaptiveAvgPool2d-100	[-1, 144, 1, 1]	0
Conv2d-101	[-1, 40, 1, 1]	5,800
ReLU-102	[-1, 40, 1, 1]	0
Conv2d-103	[-1, 144, 1, 1]	5,904
Hardsigmoid-104	[-1, 144, 1, 1]	0
SqueezeExcitation-105	[-1, 144, 14, 14]	0
Conv2d-106	[-1, 48, 14, 14]	6,912
BatchNorm2d-107	[-1, 48, 14, 14]	96
InvertedResidual-108	[-1, 48, 14, 14]	0
Conv2d-109	[-1, 288, 14, 14]	13,824
BatchNorm2d-110	[-1, 288, 14, 14]	576
Hardswish-111	[-1, 288, 14, 14]	0
Conv2d-112	[-1, 288, 7, 7]	7,200
BatchNorm2d-113	[-1, 288, 7, 7]	576
Hardswish-114	[-1, 288, 7, 7]	0
AdaptiveAvgPool2d-115	[-1, 288, 1, 1]	0
Conv2d-116	[-1, 72, 1, 1]	20,808
ReLU-117	[-1, 72, 1, 1]	0
Conv2d-118	[-1, 288, 1, 1]	21,024
Hardsigmoid-119	[-1, 288, 1, 1]	0
SqueezeExcitation-120	[-1, 288, 7, 7]	0
Conv2d-121	[-1, 96, 7, 7]	27,648
BatchNorm2d-122	[-1, 96, 7, 7]	192
InvertedResidual-123	[-1, 96, 7, 7]	0
Conv2d-124	[-1, 576, 7, 7]	55,296
BatchNorm2d-125	[-1, 576, 7, 7]	1,152
Hardswish-126	[-1, 576, 7, 7]	0
Conv2d-127	[-1, 576, 7, 7]	14,400
BatchNorm2d-128	[-1, 576, 7, 7]	1,152
Hardswish-129	[-1, 576, 7, 7]	0
AdaptiveAvgPool2d-130	[-1, 576, 1, 1]	0
Conv2d-131	[-1, 144, 1, 1]	83,088
ReLU-132	[-1, 144, 1, 1]	0
Conv2d-133	[-1, 576, 1, 1]	83,520
Hardsigmoid-134	[-1, 576, 1, 1]	0
SqueezeExcitation-135	[-1, 576, 7, 7]	0
Conv2d-136	[-1, 96, 7, 7]	55,296
BatchNorm2d-137	[-1, 96, 7, 7]	192
InvertedResidual-138	[-1, 96, 7, 7]	0
Conv2d-139	[-1, 576, 7, 7]	55,296
BatchNorm2d-140	[-1, 576, 7, 7]	1,152
Hardswish-141	[-1, 576, 7, 7]	0
Conv2d-142	[-1, 576, 7, 7]	14,400
BatchNorm2d-143	[-1, 576, 7, 7]	1,152
Hardswish-144	[-1, 576, 7, 7]	0
AdaptiveAvgPool2d-145	[-1, 576, 1, 1]	0
Conv2d-146	[-1, 144, 1, 1]	83,088
ReLU-147	[-1, 144, 1, 1]	0

```
          Conv2d-148            [-1, 576, 1, 1]          83,520
     Hardsigmoid-149            [-1, 576, 1, 1]               0
SqueezeExcitation-150            [-1, 576, 7, 7]               0
          Conv2d-151            [-1, 96, 7, 7]           55,296
     BatchNorm2d-152            [-1, 96, 7, 7]              192
InvertedResidual-153            [-1, 96, 7, 7]               0
          Conv2d-154           [-1, 576, 7, 7]           55,296
     BatchNorm2d-155           [-1, 576, 7, 7]            1,152
       Hardswish-156           [-1, 576, 7, 7]               0
AdaptiveAvgPool2d-157           [-1, 576, 1, 1]               0
          Linear-158                 [-1, 1024]         590,848
       Hardswish-159                 [-1, 1024]               0
         Dropout-160                 [-1, 1024]               0
          Linear-161                   [-1, 42]          43,050
     MobileNetV3-162                   [-1, 42]               0
================================================================
Total params: 1,560,906
Trainable params: 1,560,906
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 34.60
Params size (MB): 5.95
Estimated Total Size (MB): 41.13
```

```
FLOPS: 0.122993904 G, Params: 1.560906 M.
```

- **Original data and predicted result**



Part IV. Reference

[1] SearchingforMobileNetV3:
https://arxiv.org/pdf/1905.02244

[2] Efficient CNN 介紹(二)：MobilenetV2:
https://medium.com/ai-academy-taiwan/efficient-cnn-%E4%BB%8B%E7%B4%B9-%E4%BA%8C-mobilenetv2-7809721f0bc8

[3] MobileNetV2: Inverted Residuals and Linear Bottlenecks:
https://arxiv.org/pdf/1801.04381

[4] Squeeze-and-Excitation Networks:
https://arxiv.org/pdf/1709.01507

[5] 【深度学习】Squeeze-and-Excitation (SE) 模块优势解读:
https://www.ebaina.com/articles/140000012469