Part 1:

1-1. The name under which you submitted on Kaggle :

yuxuanz6

1-2. Best accuracy (should match your accuracy on Kaggle) :

61.0%

1-3. Table defining your final architecture similar to the image above.

Note: I've the padding = 1 and stride = 1 (for conv layers); padding = 0, stride = 2(for maxpool)

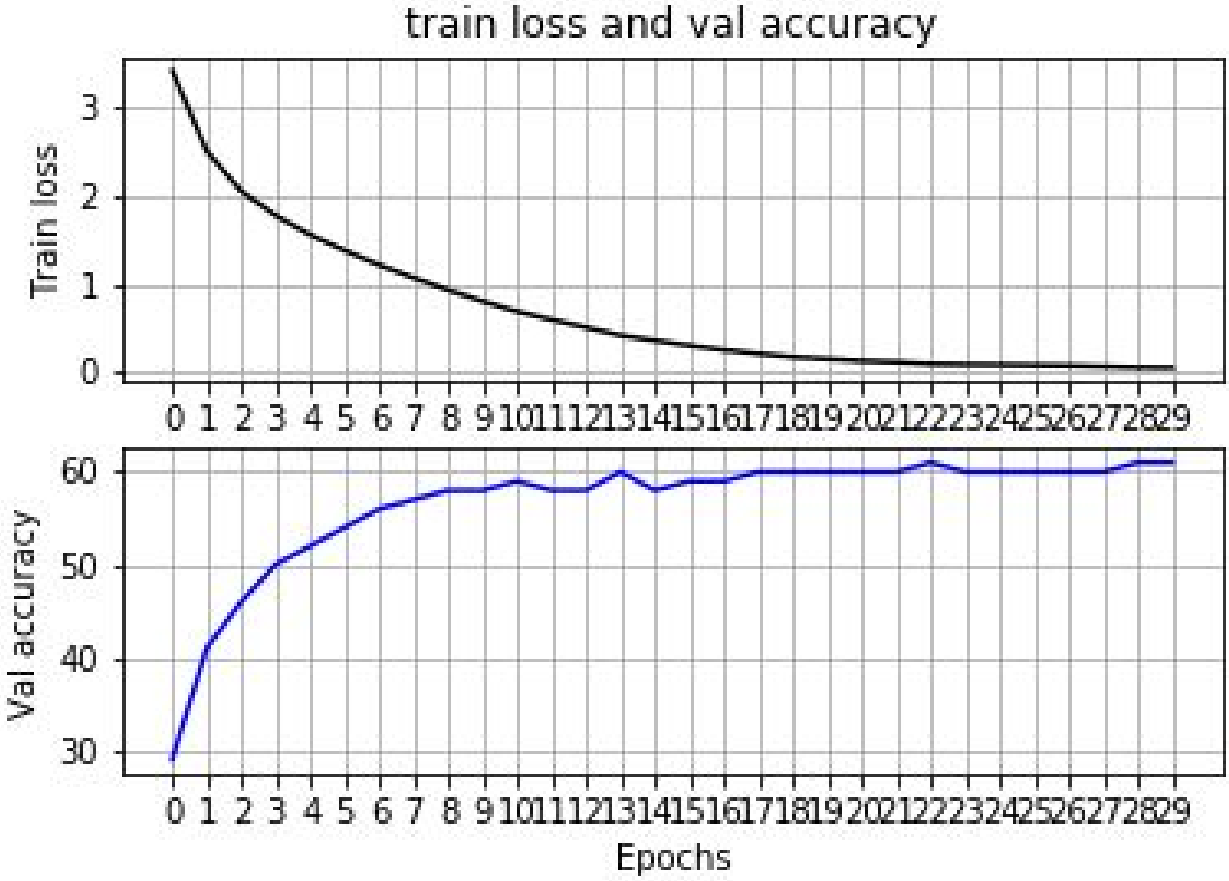
Layer No	Layer Type	Kernel Size (for conv layers)	Input|Output dimension	Input|Output Channels(for conv layers)
1	conv2d	3	32 | 32	3|64
2	batchnorm2d	-	32 | 32	64|64
3	relu	-	32 | 32	-
4	conv2d	3	32 | 32	64|64
5	batchnorm2d	-	32 | 32	64|64
6	relu	-	32 | 32	-
7	maxpool2d	2	32 | 16	-
8	conv2d	3	16 | 16	64|128
9	batchnorm2d	-	16 | 16	128|128
10	relu	-	16 | 16	-
11	conv2d	3	16 | 16	128|128

12	batchnorm2d	-	16 | 16	128|128
13	relu	-	16 | 16	-
14	maxpool2d	2	16 | 8	-
15	conv2d	3	8 | 8	128|256
16	batchnorm2d	-	8 | 8	256|256
17	relu	-	8 | 8	-
18	conv2d	3	8 | 8	256|256
19	batchnorm2d	-	8 | 8	256|256
20	relu	-	8 | 8	-
21	maxpool2d	2	8 | 4	-
22	conv2d	3	4 | 4	256|512
23	batchnorm2d	-	4 | 4	512|512
24	relu	-	4 | 4	-
25	conv2d	3	4 | 4	512|512
26	batchnorm2d	-	4 | 4	512|512
27	relu	-	4 | 4	-
28	maxpool2d	2	4 | 2	-
29	linear	-	2048|1024	-
30	batchnorm1d	-	1024|1024	-
31	relu	-	1024|1024	-
32	linear	-	1024|1024	-
33	batchnorm1d	-	1024|1024	-
34	relu	-	1024|1024	-
35	linear	-	1024|100	-

1-4. Factors which helped improve your model performance. Explain each factor in 2-3 lines.

1. Data normalization	Because the sigmoid function for the last evaluation here, I choose the first normalization method which is in the range [0,1]; Thus, the mean is about 0.5 which is std is about 0.25; Last I have: mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225]; I have also tried to use ([0.505, 0.496, 0.446], [0.259, 0.254, 0.275]),which is basically the same.
2. Data augmentation	I used two common random transforms here ---- RandomHorizontalFlip() without arguments simply randomly flip the image horizontally with probability 0.5; RandomCrop(32) indicates the output size of the image, here is 32.By applying it, I get new image data, which is horizontal flipped by the original image without get more outside image data.
3. Deeper network	I first add many conv2d layers,use same padding trick to keep the same as the original size(padding=1,stride=1), each followed with batchnorm2d, then the activation layer relu. Secondly,I use maxpool layers according to the size of input and output, each time designed to decrease by half, use the most common (padding=0,stride=2). The performance enhance from 13% to around 41%. Finally, I add fc layer that is one linear layer followed by batchnorm1d layer, then the activation layer relu, which enhances the performance from 13% to around 55%. After adjusting the inputs/outputs dimension and the numbers of layers, finally it reaches around 61%.
4. Normalization layers	The idea is to normalise the inputs of each layer that they have a mean output activation of zero and standard deviation of one. This is analogous to how the inputs to networks are standardised. By using batch normalisation, we could do much higher learning rates, increasing the speed at which networks train. It's also robust (in train) to the scale & shift of input data and scale of update decreases while training
5. Early stopping	In machine learning, too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model. Early stopping is a method to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold out validation dataset. I set my epoch = 30 because the accuracy will not enhance anymore in most cases after 25.

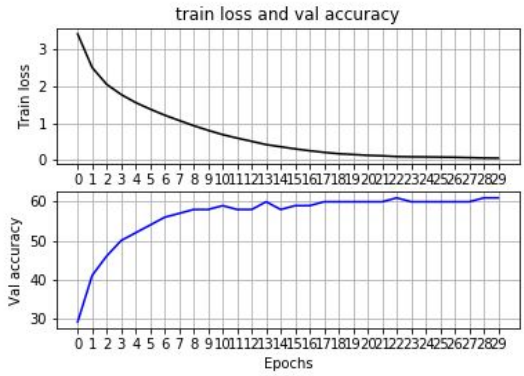
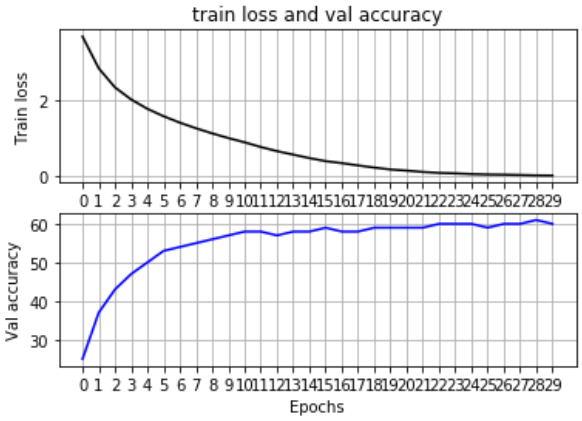
1-5. Final architecture's plot for training loss and validation accuracy. This would have been auto-generated by the notebook.



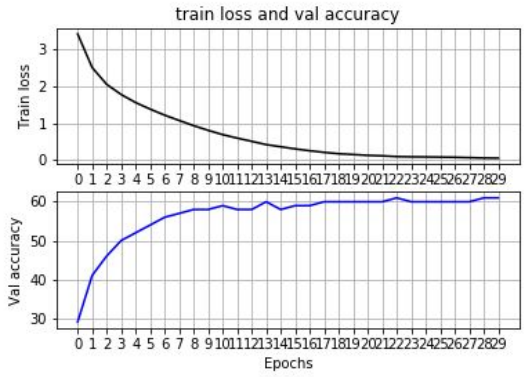
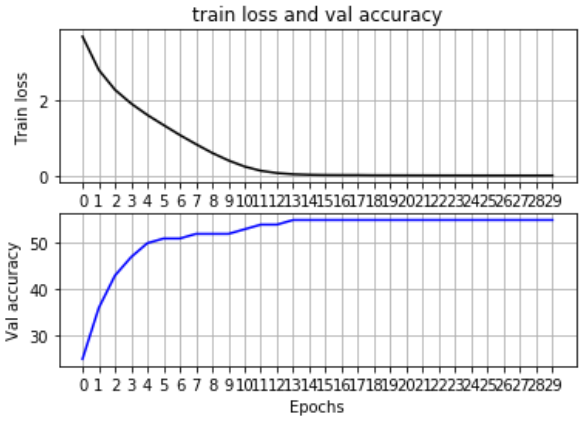
1-6. Ablation study to validate the above choices, i.e., a comparison of performance for two variants of a model, one with and one without a certain feature or implementation choice.

Comparison 1

With **Data normalization**	Without **Data normalization**

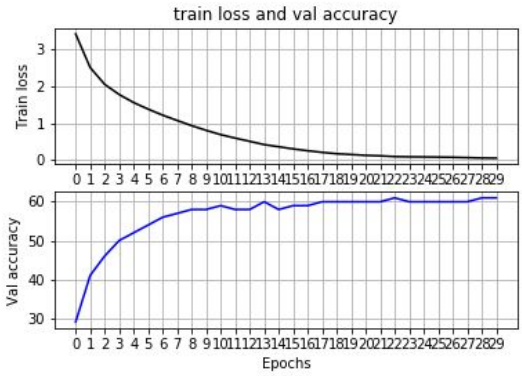
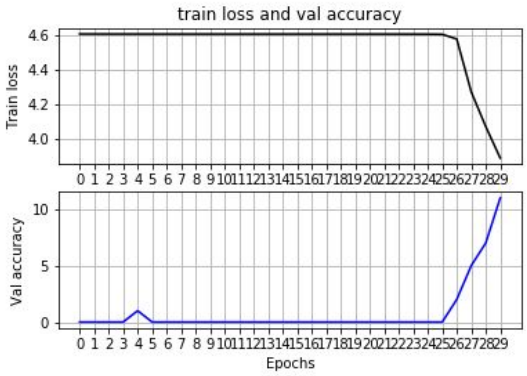
Keep the final accuracy 61% 	Keep the final accuracy 60% Which means I might not normalization well 

Comparison 2

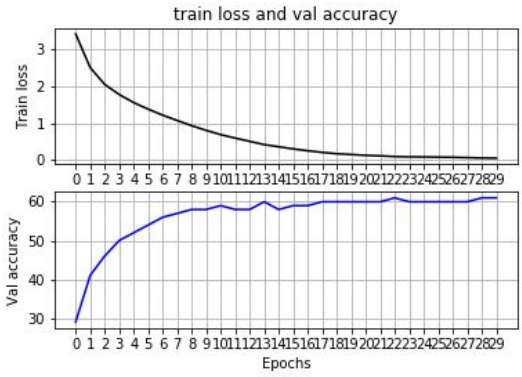
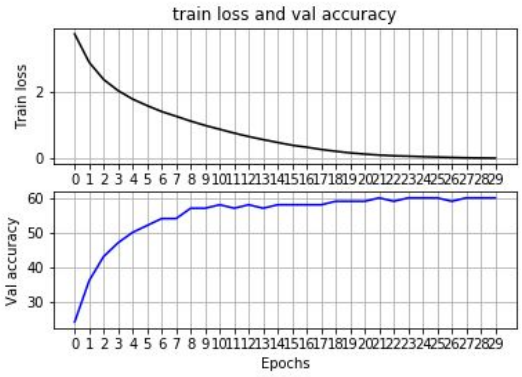
With **Data augmentation** Keep the final accuracy 61% 	Without **Data augmentation** Keep the final accuracy 55 % 

Comparison 3

With **Normalization layers** Keep the final accuracy 61%	Without **Normalization layers** Keep accuracy 11% ,can't do well job, we might need to change the epochs to be greater than 30, from epoch = 26 to 30, the accuracy increases from 0 to 11. If we increases the epoch, we might also get the accuracy to be 61% eventually. However, the running time must be longer than the

	normalization layers design. 

Comparison 4

With **Data normalization** mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225]; Keep the final accuracy 61% 	With **Data normalization** mean =[0.505, 0.496, 0.446], std = [0.259, 0.254, 0.275]; Keep the final accuracy 60%, which means this data normalization doesn't change too much towards the accuracy result 

1-7. Explanation of any extra credit features if attempted.

1.First, I've tried the dropout method and flatten in the following in the fc layers and can speeding up; However, accuracy doesn't enhance(around 59%). Here is the part of the code:

```
class Flatten(torch.nn.Module):
```

```
        def forward(self, x):
          return x.view(x.size()[0], -1)
```

```
1.torch.nn.Dropout()
```

```
2.torch.nn.Dropout(0.5)
```

```
3.Flatten()
```

2. Second, set the epoch = 30, others tries like sets the learning rate from 0.005 to 0.05

Lr = 0.001 and epoch = 30 => accuracy = 61%;

Lr = 0.005 and epoch = 30 => accuracy = 61%;

Lr = 0.01 and epoch = 30 => accuracy = 59%;

Lr = 0.05 and epoch = 30 => accuracy = 59%;

Thus, the original learning rate is suitable and 0.001 will be another choice.

3. Third, I've studied different optimizer choices except from sgd momentum. The following are:

Optimizer Types:	Accuracy
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)	61%
optimizer = optim.Adam(net.parameters(), lr=1e-3)	59%
optimizer = optim.Adagrad(net.parameters(), lr=1e-1, lr_decay=1e-3)	41 %
optimizer = optim.Adamax(net.parameters(), lr=1e-1,weight_decay=1e-2)	< 10%
optimizer = optim.RMSprop(net.parameters(), lr=1e-3, lr_decay=1e-2)	20%
optimizer = optim.Adadelta(net.parameters(), lr=1e-3, rho=0.95)	39%
optimizer = optim.Rprop(net.parameters(), lr=1e-3, etas=(0.6, 1.1))	< 10%

optimizer = optim.ASGD(net.parameters(), lr=1e-3, alpha=0.8)	50%

Here, if study in details about those parameters, we can get better results. Due to the limitation of study now, I'm not going into too much about those optimizers now. One optimizer should be better than sgd is Adagrad, just don't know how to set the parameters properly. Adam might also be a better one as well.

Part 2:

2-1.Report the train and test accuracy achieved by using the ResNet as a fixed feature extractor vs. fine-tuning the whole network.

After the several tryings, I find the NUM_EOCHS >=30 and LEARNING_RATE = 0.01 is good to get high train accuracy and test accuracy. Thus, I start to use NUM_EOCHS = 30 and 50 and LEARNING_RATE = 0.01, change the BATCH_SIZE from 8 to 16,24,32,40,80 and compare the fixed feature extractor and the fine-tuning the whole network. The best performance and its setting is:

```
Train Accuracy 0.9353
Test Accuracy 0.9317
```

NUM_EPOCHS = 50
LEARNING_RATE = 0.01
BATCH_SIZE = 80
RESNET_LAST_ONLY = False

Whatever, in most cases, the accuracy of fixed feature extractor is worse than fine-tuning the whole network. However, the running speed of fixed feature extractor is faster than fine-tuning the whole network.

Note: in the part2 - extra, I'm able to get higher accuracy than this one by using different net.

Settings and both train and test accuracy are above 80%(Except the first one):

hyperparameter settings	fixed feature extractor RESNET_LAST_ONLY = True	fine-tuning the whole network RESNET_LAST_ONLY = False
NUM_EPOCHS = 30 LEARNING_RATE = 0.01 BATCH_SIZE = 16	`Train Accuracy 0.7697` `Test Accuracy 0.8020`	`Train Accuracy 0.8300` `Test Accuracy 0.8417`
NUM_EPOCHS = 50	`Train Accuracy 0.8063`	`Train Accuracy 0.8747`

LEARNING_RATE = 0.01 BATCH_SIZE = 16	Test Accuracy 0.8630	Test Accuracy 0.8937
NUM_EPOCHS = 30 LEARNING_RATE = 0.01 BATCH_SIZE = 24	Train Accuracy 0.8107 Test Accuracy 0.8008	Train Accuracy 0.8520 Test Accuracy 0.8503
NUM_EPOCHS = 50 LEARNING_RATE = 0.01 BATCH_SIZE = 24	Train Accuracy 0.8273 Test Accuracy 0.8515	Train Accuracy 0.8907 Test Accuracy 0.8970
NUM_EPOCHS = 30 LEARNING_RATE = 0.01 BATCH_SIZE = 32	Train Accuracy 0.8093 Test Accuracy 0.8027	Train Accuracy 0.8910 Test Accuracy 0.8891
NUM_EPOCHS = 50 LEARNING_RATE = 0.01 BATCH_SIZE = 32	Train Accuracy 0.8417 Test Accuracy 0.8678	Train Accuracy 0.9223 Test Accuracy 0.9126
NUM_EPOCHS = 30 LEARNING_RATE = 0.01 BATCH_SIZE = 40	Train Accuracy 0.8693 Test Accuracy 0.8625	Train Accuracy 0.8970 Test Accuracy 0.8992
NUM_EPOCHS = 50 LEARNING_RATE = 0.01 BATCH_SIZE = 40	Train Accuracy 0.8717 Test Accuracy 0.8838	Train Accuracy 0.9243 Test Accuracy 0.9199
NUM_EPOCHS = 50 LEARNING_RATE = 0.01 BATCH_SIZE = 80	Train Accuracy 0.9124 Test Accuracy 0.9002	Train Accuracy 0.9353 Test Accuracy 0.9317

The following are initial tryings, indicates how I get into the above ones

Settings and train accuracy is below 80%:

hyperparameter settings	fixed feature extractor RESNET_LAST_ONLY = True	fine-tuning the whole network RESNET_LAST_ONLY = False
NUM_EPOCHS = 10 LEARNING_RATE = 0.0001 BATCH_SIZE = 8	Train Accuracy 0.0853 Test Accuracy 0.1227	Train Accuracy 0.2193 Test Accuracy 0.3272
NUM_EPOCHS = 30 LEARNING_RATE = 0.0001	Train Accuracy 0.3297 Test Accuracy 0.4431	Train Accuracy 0.5527 Test Accuracy 0.6634

BATCH_SIZE = 8		
NUM_EPOCHS = 50 LEARNING_RATE = 0.0001 BATCH_SIZE = 8	Train Accuracy 0.4540 Test Accuracy 0.5366	Train Accuracy 0.6933 Test Accuracy 0.7715
NUM_EPOCHS = 10 LEARNING_RATE = 0.001 BATCH_SIZE = 8	Train Accuracy 0.4613 Test Accuracy 0.6063	Train Accuracy 0.6983 Test Accuracy 0.8079
NUM_EPOCHS = 10 LEARNING_RATE = 0.01 BATCH_SIZE = 8	Train Accuracy 0.5283 Test Accuracy 0.7209	Train Accuracy 0.0743 Test Accuracy 0.1191
NUM_EPOCHS = 30 LEARNING_RATE = 0.01 BATCH_SIZE = 8	Train Accuracy 0.6347 Test Accuracy 0.7817	Train Accuracy 0.5150 Test Accuracy 0.6221
NUM_EPOCHS = 50 LEARNING_RATE = 0.01 BATCH_SIZE = 8	Train Accuracy 0.6950 Test Accuracy 0.8147	Train Accuracy 0.7347 Test Accuracy 0.8145

2-2.Report any hyperparameter settings you used (batch_size, learning_rate, resnet_last_only, num_epochs).

hyperparameter settings	Train Accuracy	Test Accuracy
NUM_EPOCHS = 10 LEARNING_RATE = 0.0001 BATCH_SIZE = 8 RESNET_LAST_ONLY = False	0.2193	0.3272
NUM_EPOCHS = 30 LEARNING_RATE = 0.0001 BATCH_SIZE = 8 RESNET_LAST_ONLY = False	0.5527	0.6634
NUM_EPOCHS = 50 LEARNING_RATE = 0.0001 BATCH_SIZE = 8 RESNET_LAST_ONLY = False	0.6933	0.7715

NUM_EPOCHS = 10 LEARNING_RATE = 0.001 BATCH_SIZE = 8 RESNET_LAST_ONLY = False	0.6983	0.8079
NUM_EPOCHS = 10 LEARNING_RATE = 0.01 BATCH_SIZE = 8 RESNET_LAST_ONLY = False	0.0743	0.1191
NUM_EPOCHS = 30 LEARNING_RATE = 0.01 BATCH_SIZE = 8 RESNET_LAST_ONLY = False	0.5150	0.6221
NUM_EPOCHS = 50 LEARNING_RATE = 0.01 BATCH_SIZE = 8 RESNET_LAST_ONLY = False	0.7347	0.8145
Note: now the following settings will bring both the training and testing accuracy above 80%		
NUM_EPOCHS = 30 LEARNING_RATE = 0.01 BATCH_SIZE = 16 RESNET_LAST_ONLY = False	0.8300	0.8417
NUM_EPOCHS = 50 LEARNING_RATE = 0.01 BATCH_SIZE = 16 RESNET_LAST_ONLY = False	0.8747	0.8937
NUM_EPOCHS = 30 LEARNING_RATE = 0.01 BATCH_SIZE = 24 RESNET_LAST_ONLY = False	0.8520	0.8503
NUM_EPOCHS = 50 LEARNING_RATE = 0.01 BATCH_SIZE = 24 RESNET_LAST_ONLY = False	0.8907	0.8970
NUM_EPOCHS = 30 LEARNING_RATE = 0.01 BATCH_SIZE = 32 RESNET_LAST_ONLY = False	0.8910	0.8892

NUM_EPOCHS = 50 LEARNING_RATE = 0.01 BATCH_SIZE = 32 RESNET_LAST_ONLY = False	0.9223	0.9126
NUM_EPOCHS = 30 LEARNING_RATE = 0.01 BATCH_SIZE = 40 RESNET_LAST_ONLY = False	0.8970	0.8992
NUM_EPOCHS = 50 LEARNING_RATE = 0.01 BATCH_SIZE = 40 RESNET_LAST_ONLY = False	0.9243	0.9199
NUM_EPOCHS = 50 LEARNING_RATE = 0.01 BATCH_SIZE = 80 RESNET_LAST_ONLY = False	0.9353	0.9317

2-3 Extra Part

I'm using resnet50 and resnet101 here, note resnet151 will run out of memory, though they have better performance. Any model whose layer are above 100, I will not try due to memory problem. Another choice for me is Inception v3, and input size here is 229 x 229, however, will cause softmax error.

Net's name	hyperparameter settings	Train Accuracy	Test Accuracy
resnet50	NUM_EPOCHS = 50 LEARNING_RATE = 0.01 BATCH_SIZE = 80 RESNET_LAST_ONLY = False	0.9423	0.9311
resnet101	NUM_EPOCHS = 50 LEARNING_RATE = 0.01 BATCH_SIZE = 80 RESNET_LAST_ONLY = False	0.9483	0.9411