Question 1:

Download the Tiny Imagenet data which has 200 classes and each class has 500 images, 50 validation images and 50 test images. Firstly we changed the size of images to 32X32 format with these code:

```
import os
directory="./tiny-imagenet-200/train/"
all_folder=os.listdir(directory)
all_folder_real=[x for x in all_folder if x[0]=='n']

for sr in all_folder_real:
    raw="python image_resizer_imagent.py -i ./train/"+sr+"  -o ./data -s 32 -a box -r -j 1
    print(raw)
```

```
python image_resizer_imagent.py -i ./train/n01443537  -o ./data -s 32 -a box -r -j 1
0
python image_resizer_imagent.py -i ./train/n01629819  -o ./data -s 32 -a box -r -j 1
0
python image_resizer_imagent.py -i ./train/n01641577  -o ./data -s 32 -a box -r -j 1
0
python image_resizer_imagent.py -i ./train/n01644900  -o ./data -s 32 -a box -r -j 1
0
python image_resizer_imagent.py -i ./train/n01698640  -o ./data -s 32 -a box -r -j 1
0
python image_resizer_imagent.py -i ./train/n01742172  -o ./data -s 32 -a box -r -j 1
```

Figure 1: transform the images to 32*32*3 pattern

Then split the dataset into x_train, y_train, x_validation, y_validation, x_test. Next we used "Pikle" to upload the dataset to Colab, however due to the dataset is too large to upload once, we splite the x_training data into 5 parts to upload like this:

```
Upload widget is only available when the cell has been executed in the current browser session. Please
rerun this cell to enable.

Saving y_valid.pkl to y_valid.pkl
Saving x_valid.pkl to x_valid.pkl
Saving x_test.pkl to x_test.pkl
Saving y_train.pkl to y_train.pkl
Saving x_train_p5.pkl to x_train_p5.pkl
Saving x_train_p4.pkl to x_train_p4.pkl
Saving x_train_p3.pkl to x_train_p3.pkl
Saving x_train_p2.pkl to x_train_p2.pkl
Saving x_train_p1.pkl to x_train_p1.pkl
```

Figure 2: upload data to Colab with pikle

We make a for loop to obtain the optimal model for the dataset. As for the parameters of Conv2D, we mainly adjusted the parameters of "filters" and "drop_out", and "kernel size" to run the model to check the training and validation accuracy as following graph. Because of the dataset is 32*32*3 images, so the padding, activation function, stride, pooling size should be the same values which are "SAME", "RELU", 1 and 2*2. If we set the value of stride and pooling size larger, it is easy to lose some information of the images. And for the tiny images, we should detect the images bound to get more features and information. Besides we choose "categorical_crossentropy" to be loss function, "adam" to be optimizer, other parameters to be default value.

```python
for drop_out in drop_outs:
    for filter_base in filter_bases:
        for kernel_size in kernel_sizes:
            mid_filter=filter_base*2
            high_filter=filter_base*4
            print(drop_out, filter_base, mid_filter, high_filter)
            model = Sequential()
            model.add(Conv2D(filter_base, kernel_size = kernel_size, padding='same', activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 3)))
            model.add(MaxPooling2D(pool_size=(2,2)))
            model.add(BatchNormalization())
            model.add(Conv2D(mid_filter, kernel_size=kernel_size, padding='same', activation='relu'))
            model.add(MaxPooling2D(pool_size=(2,2)))
            model.add(BatchNormalization())
            model.add(Conv2D(mid_filter, kernel_size=kernel_size, padding='same', activation='relu'))
            model.add(MaxPooling2D(pool_size=(2,2)))
            model.add(BatchNormalization())
            model.add(Conv2D(mid_filter, kernel_size=kernel_size, padding='same', activation='relu'))
            model.add(MaxPooling2D(pool_size=(2,2)))
            model.add(BatchNormalization())
            model.add(Dropout(drop_out))
            model.add(Flatten())
            model.add(Dense(256, activation='relu'))
            model.add(Dropout(drop_out))
            model.add(Dense(200, activation = 'softmax'))

            model.compile(loss='categorical_crossentropy', optimizer='adam', metrics = ['accuracy'])
            print(model.summary())
            history= model.fit(x_train, y_train2, shuffle=True, batch_size = 128, epochs = 15, verbose = 1, validation_data=(x_valid, y_valid2)
```

Figure 3: model sample

So we get the following table for the loop:

	parameters	state1	state2	state3	state4	state5	state6	state7	state8	state9	state10
	epochs	30	30	30	30	30	30	30	30	30	30
	batch_size	128	128	128	128	128	128	128	128	128	128
1	No. conv	4	4	6	6	4	6	8	4	4	4
2	filters	128	128	128	128	32	32	32	128	256	128
		128	256	128	128	32	32	32	128	256	128
		256	256	256	256	32	64	32	256	256	256
		256	256	256	256	256	64	32	256	256	256
				256	512		256	64			
				256	512		256	64			
								256			
								256			
3	kernal_size	3	3	3	4	4	4	4	4	5	5
4	strides	1	1	1	1	1	1	1	1	1	1
5	drop	0.3	0.3	0.3	0.3	0.4	0.4	0.4	0.5	0.5	0.5
5	padding	SAME	VALID	SAME	SAME	SAME	SAME	SAME	SAME	SAME	SAME
6	activation	relu	relu	relu	relu	relu	relu	relu	relu	relu	relu
	pooling size	2	2	2	2	2	2	2	2	2	2
7	dilation_rate	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)
8	ernal_initialize	orot_unifor	orot_unifor	orot_unifor	orot_unifor	orot_unifor	orot_unifor	orot_unifor	orot_unifor	orot_unifor	orot_uniform
9	bias_initializer	zeros	zeros	zeros	zeros	zeros	zeros	zeros	zeros	zeros	zeros
10	ernel_regulariz	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE
11	bias_regularize	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE
12	tivity_regulariz	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE
14	train_loss	1.6247	2.3562	1.3241	1.0637	2.6219	0.5076	3.0011	2.033	1.3513	1.8562
15	val_loss	3.159	3.0464	3.5281	3.7567	2.9611	4.6028	3.2348	3.0811	3.6388	3.0865
16	diff_loss	1.5343	0.6902	2.204	2.693	0.3392	4.0952	0.2337	1.0481	2.2875	1.2303
17	train_acc	0.5572	0.4131	0.624	0.6854	0.3618	0.8426	0.2927	0.469	0.6145	0.5028
18	val_acc	0.3166	0.3062	0.2898	0.3065	0.3066	0.2993	0.2644	0.3156	0.2863	0.3323
19	diff_acc	0.2406	0.1069	0.3342	0.3789	0.0552	0.5433	0.0283	0.1534	0.3282	0.1705

Figure 4: experiments output

During the computing process, we know the models can be convergent less than 15 epochs. So we lastly tune the parameters with 15 epochs to avoid to waste computing time. And then we use the final for loop get the relative optimal model which has 4 layers, two sense layers, drop_out being 0.3 the filters being 128, 128, 256,256, other parameters are the default value or the original values mentioned above. Following is the corresponding training & validation accuracy graph and loss graph:

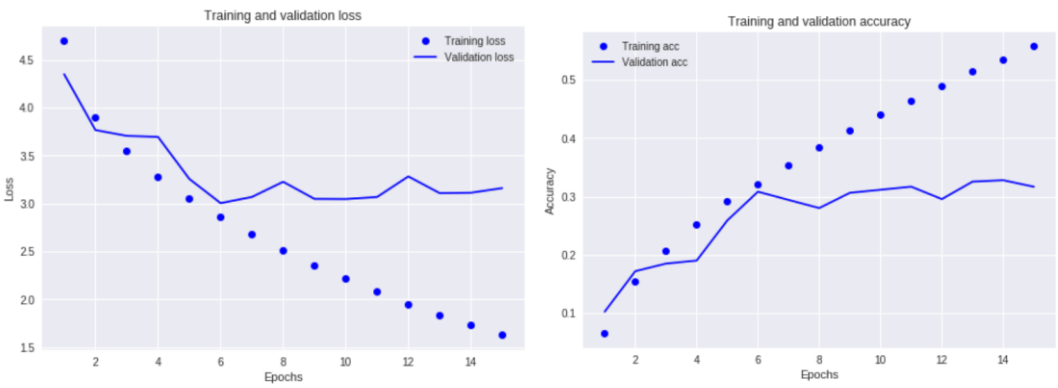

Figure 5: accuracy and loss graph of optimal model

The loss and accuracy values are as following:

```
100000/100000 [==============================] - 24s 240us/step
Train loss: 1.1080415958976746
Train accuracy: 0.71935
10000/10000 [==============================] - 2s 244us/step
Validation loss: 3.159041582107544
Validation accuracy: 0.3166
```

Figure 6: accuracy and loss values

As for the test data, we can not upload to the StandFord test web without `<your SUNetID>.txt`, so we reserve y_test data into a new file which can be test with the real test values.

problem1 output--test_label_predicted.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
test_3671.JPEG n03992509test_5200.JPEG n02699494test_6709.JPEG n04501370test_1066.JPEG
n02927161test_8724.JPEG n03837869test_7417.JPEG n09332890test_8042.JPEG n03126707
test_7371.JPEG n03992509test_1700.JPEG n02841315test_985.JPEG n02321529test_5566.JPEG
n04133789test_2209.JPEG n04501370test_3117.JPEG n02321529test_4678.JPEG n04417672
test_5572.JPEG n02837789test_3103.JPEG n02236044test_8056.JPEG n07720875test_749.JPEG
n02509815test_7365.JPEG n02769748test_1714.JPEG n02814860test_991.JPEG n02481823
test_9348.JPEG n02892201test_1072.JPEG n02125311test_8730.JPEG n04179913test_7403.JPEG
n01984695test_3665.JPEG n02927161test_5214.JPEG n01944390test_6735.JPEG n07747607
test_9406.JPEG n02948072test_8718.JPEG n02231487test_4122.JPEG n03891332test_2553.JPEG
n02892201test 3895.JPEG n02948072test 2235.JPEG n01768244test 4644.JPEG n04532670
```

Figure 7: x_test output waiting to be test with real labels

Question 2:

1. Input data:

As question 1, we use the dataset whch has been splitted into x_train, y_train, x_validation, y_validation, x_test to train an auto model.

```
all_train_images=os.listdir("./data/box/images")
#print(len(all_train_images))
labels=[x.split("_")[0] for x in all_train_images]
labels_my=[int(x.split("_")[1].split('.png')[0]) for x in all_train_images]
labels_my

# trail all, delete

filt=pd.DataFrame({'image':all_train_images,'label':labels,'num':labels_my})
filt
all_train_images=filt['image'].tolist()
labels=filt['label'].tolist()

with open('train_label.csv', 'w') as train_csv:
    fieldnames = ['File Name', 'Label']
    writer = csv.DictWriter(train_csv, fieldnames=fieldnames)
    writer.writeheader()
    for i in range(len(all_train_images)):
        writer.writerow({'File Name': all_train_images[i], 'Label':labels[i]})
    train_csv.close()

x_train, y_train = load_image_dataset(csv_file_path="train_label.csv",
                                      images_path="./data/box/images")
print(x_train.shape)
print(y_train.shape)
```

2. Then we fit a model with ImageClassifier and adjust the parameters for the model. This classifier has three kinds of parameters: (1) Searcher: MAX_MODEL_NUM, BETA, KERNEL_LAMBDA, T_MIN, N_NEIGHBOURS, MAX_MODEL_SIZE, (2) Model Defaults: DENSE_DROPOUT_RATE, CONV_DROPOUT_RATE, CONV_BLOCK_DISTANCE, DENSE_BLOCK_DISTANCE, MODEL_LEN, MODEL_WIDTH, (3) model trainer: DATA_AUGMENTATION, MAX_NO_IMPROVEMENT_NUM. MAX_ITER_NUM, MIN_LOSS_DEC.

But for our case, we just need to adjust a part of the parameters of max_no_impr-ovement_num, and max_iter_num, and the time limit.

```
TRAINING_TIMES = [
    60 * 60 * 6,  # 6 hour
    60 * 60 * 2,  # 2 hours
    60 * 60 * 3,  # 3 hours
    60 * 60 * 4,  # 4 hours
]
```

```
results=[]
for seconds in TRAINING_TIMES:
    print("[INFO] training model for {} seconds max...".format(seconds))
    clf = ImageClassifier(verbose=True,augment=True, searcher_args={'trainer_args':{'max_no_improvement_num':4}})
    clf.fit(x_train, y_train, time_limit=seconds)
    fitted=clf.final_fit(x_train, y_train,x_valid, y_valid, retrain=True)
    print(clf.summary())
    # evaluate the Auto-Keras model
    score_train = clf.evaluate(x_train, y_train)
    score_valid = clf.evaluate(x_valid, y_valid)
    y_predict = clf.predict(x_test)

    print(score_train,score_valid)
    #results.append([clf.copy(),seconds,score_train,score_valid,score_test])
results
```

So we run the auto model with the most iteration number being 15, which means the epochs for each model is at most 15. And if one spcific model doesn't change the maccuracy four consecutive times, than move to the next model. And for the whole auto running process, we should set max running time to get the optimal model in a limited time due to the realistic situation. The "clf.fit" trained out a relative optimal model with optimal structure, and then use "clf.final_fit" to further train better parameters of the model based on the optimal structure.
At last, auto keras output four model, and the best model with accuracy like this:

```
Saving model.
+--------------------------------------------------------------------------+
|        Model ID        |          Loss          |      Metric Value      |
+--------------------------------------------------------------------------+
|           2            |   15.403569221496582   |         0.2125         |
+--------------------------------------------------------------------------+

+----------------------------------------------+
|              Training model 3                |
+----------------------------------------------+

No loss decrease after 4 epochs.

Saving model.
+--------------------------------------------------------------------------+
|        Model ID        |          Loss          |      Metric Value      |
+--------------------------------------------------------------------------+
|           3            |   15.228576719760895   |         0.2525         |
+--------------------------------------------------------------------------+

+----------------------------------------------+
|              Training model 4                |
+----------------------------------------------+

No loss decrease after 4 epochs.

Time is out.

No loss decrease after 30 epochs.

0.43475 0.275
```

Traing_accuracy: 0.43474, validation_accuracy: 0.275, which is a little worse than problem 1 we got. Due to the computing time is much more than self-build model, autokeras maybe not the first choice to build models.