Report of Experiment3&4

1. **Date Processing**

   First, we use the dataset: tiny imagenet with 200 classes to train, validation and test. The shape of this dataset is 64, which is big so that would cost huge time, so we resized their shape as 32x32. We use the following code to reshape the dataset.

```
for i in range(0,len(sub_folders)):
    i_f = input_folder+str(sub_folders[i])+"/images/"
    o_f = output_folder+str(sub_folders[i])
    # os.mkdir(output_folder+str(sub_folders[i]))
    command = "python image_resizer_imagent.py -i "+i_f+" -o "+o_f+" -s 32 -a box -j 10"

    print(command)
    os.system(command)
```

Code Snippet 1

   After resizing the data, we use following codes to load image data and separate them into training data, validation data. And I save loaded data as .npy file to save time.

```
# load tiny imagenet data
data_path = '/content/tiny-imagenet-200-32/'
train_data_path = '/content/tiny-imagenet-200-32/train/'
test_data_path = '/content/tiny-imagenet-200-32/test/box/'
val_data_path = '/content/tiny-imagenet-200-32/val/box/'

train_data = []
train_label = []
label_dict = dict()

val_data = []
val_label = []

train_exist=os.path.exists('/content/drive/ColabWorkSpaces/INFO7374/Assignment2/imagenet_data/tr
if train_exist:
  print('exist: load the data')
  train_data=np.load('/content/drive/ColabWorkSpaces/INFO7374/Assignment2/imagenet_data/train_da
  train_label=np.load('/content/drive/ColabWorkSpaces/INFO7374/Assignment2/imagenet_data/train_l
else:
  print('does not exist')
  classes = os.listdir(train_data_path)
  for i in range(len(classes)):
      label_dict[classes[i]]=i
      path = train_data_path+classes[i]+'/images/'
      image_names = os.listdir(path)
      for name in image_names:
        image_path = path+name
        x = image.load_img(image_path)
        x = image.img_to_array(x)
        train_data.append(x)
        train_label.append(i)
        print(image_path)

  train_data = np.asarray(train_data)
  train_label=np.asarray(train_label)
  np.save('/content/drive/ColabWorkSpaces/INFO7374/Assignment2/imagenet_data/train_data.npy',tra
  np.save('/content/drive/ColabWorkSpaces/INFO7374/Assignment2/imagenet_data/train_label.npy',tr
```

Code Snippet 2

2. **Utility and Early Stopping**

   In order to save the training history and the result, I wrote a sort of functions to save my work.

```
def load_train_model(model, x_train, y_train, x_test, y_test, epochs, batch_size):
    history = model.fit(x_train, y_train,
                        batch_size=batch_size,
                        epochs=epochs,
                        validation_data=(x_test, y_test),
                        shuffle=True)

# Score trained model.
    scores = model.evaluate(x_test, y_test, verbose=1)
    print('Test loss:', scores[0])
    print('Test accuracy:', scores[1])

def write_result(exper_result, path):
    with open(path, 'wb') as f:  # Python 3: open(..., 'wb')
        pickle.dump(exper_result, f)
        f.close()

def load_result(path):
    # Getting back the objects:
    with open(path,'rb') as f:  # Python 3: open(..., 'rb')
        exper_result = pickle.load(f)
        f.close()
        return exper_result
```

Code Snippet 3

Then I used early stopping to avoid my neural network from overfitting. In this function, I chose val_acc to observe. If val_acc is not improved, the model would not be trained. Here is the code of early stopping

```
early_stopping = EarlyStopping(monitor='val_acc', min_delta=0, patience=10, verbose=0, mode='auto', baseline=None, restore_best_weights=False)
history = model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(x_test, y_test),
              shuffle=True,
              callbacks=[early_stopping])
```

Code Snippet4

3. Experiment3

I used Alexnet as my training model and did eight different experiments to observe how different combinations of parameters influence the result. The following code is the function of Alexnet model.

```
def alexnet(x_train, y_train, x_test, y_test, epochs, batch_size, conv_filters_list, dense_filters_list, conv_strides, pooling_strides, optimizer, drop_out_rate):
#   build the model
    model = Sequential()

    # 1st Convolutional Layer
    model.add(Conv2D(filters=conv_filters_list[0], kernel_size=(5,5), strides=conv_strides, input_shape=(32, 32, 3), padding='valid', data_format='channels_last', activation='relu', kernel_initializer='uniform'))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=pooling_strides))

    # 2st Convolutional Layer
    model.add(Conv2D(conv_filters_list[1], (4, 4), strides=conv_strides, padding='same', activation='relu', kernel_initializer='uniform'))
    model.add(MaxPooling2D(pool_size=(5, 5), strides=pooling_strides))

    # 3st Convolutional Layer
    model.add(Conv2D(conv_filters_list[2], (3, 3), strides=conv_strides, padding='same', activation='relu', kernel_initializer='uniform'))

    # 4st Convolutional Layer
    model.add(Conv2D(conv_filters_list[3], (3, 3), strides=conv_strides, padding='same', activation='relu', kernel_initializer='uniform'))

    # 5st Convolutional Layer
    model.add(Conv2D(conv_filters_list[4], (3, 3), strides=conv_strides, padding='same', activation='relu', kernel_initializer='uniform'))

    model.add(MaxPooling2D(pool_size=(3, 3), strides=pooling_strides))
    model.add(Flatten())

    # 1st Dense Layer
    model.add(Dense(dense_filters_list[0], activation='relu'))
    model.add(Dropout(drop_out_rate))

    # 2st Dense Layer
    model.add(Dense(dense_filters_list[1], activation='relu'))
    model.add(Dropout(drop_out_rate))

    # 3st Dense Layer
    model.add(Dense(200, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    model.summary()

    early_stopping = EarlyStopping(monitor='val_acc', min_delta=0, patience=10, verbose=0, mode='auto', baseline=None, restore_best_weights=False)
    history = model.fit(x_train, y_train,
                  batch_size=batch_size,
                  epochs=epochs,
                  validation_data=(x_test, y_test),
                  shuffle=True,
                  callbacks=[early_stopping])

# Score trained model.
    scores = model.evaluate(x_test, y_test, verbose=1)
    print('Test loss:', scores[0])
    print('Test accuracy:', scores[1])

#   return history
    return (model,history, scores[0],scores[1])
```

Code Snippet5

I used Excel to make the following comprehensive table which summarizes all experiments.

							AlexNet				
Experiment No.	Epoch	Batch size	conv filters list	dense filters list	Optimizer	Early Stop Epoch	Cost time	Loss	Acc	Test loss	Test accuracy
1	100	256	[96,256,384,384,256]	[4096,4096]	sgd	100	6000	2.1851	0.4552	3.2847	0.2995
2	100	512	[96,256,384,384,256]	[4096,4096]	sgd	100	5300	3.3241	0.2511	3.3812618	0.2445
3	100	1024	[96,256,384,384,256]	[4096,4096]	sgd	100	5100	4.0581	0.1354	3.969670265	0.1498
4	200	256	[96,256,384,384,256]	[4096,4096]	sgd	80	4880	2.6776	0.3635	3.2614	0.2793
5	100	1024	[96,256,384,384,256]	[4096,4096]	adam	25	1275	1.5655	0.5742	4.622	0.2079
6	100	2048	[96,256,384,384,256]	[4096,4096]	adam	28	1456	1.2371	0.6501	5.1944	0.2049
7	200	256	[192,512,768,768,512]	[4096,4096]	sgd	78	10842	1.6047	0.5751	3.5932	0.2642
8	200	512	[192,512,768,768,512]	[4096,4096]	sgd	106	14628	2.5551	0.3883	3.3896	0.2624

Table 1

Through the table, we could find that the model of experiment 1 is the beset model. Although the training accuracy of this model is not the highest, the test

accuracy is the best. Comparing Experiment1,2,3, we could find that the increase of batch size would result in worse test accuracy. The reason of this observation is that a larger would result in a significant degradation in the quality of the model and the generalization ability of the model [1]. Comparing Experiment3,5, we could find that adam optimizer could improve the accuracy of the model when the batch size is larger. In Experiment4,7, I doubled the number of filter, the model become deeper, but the result is not improved and cost more time.

4. **Experiment4**

I used Resnet50 as my training model and did six different experiments to observe how different combinations of parameters influence the result. The following code is the function of Resnet50 model.

```
def resnet50(x_train, y_train, x_test, y_test, epochs, batch_size, optimizer, pooling):

  model = ResNet50(input_shape=(32, 32, 3), include_top=True, weights=None, pooling=pooling, classes=200)

  model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
  model.summary()

  early_stopping = EarlyStopping(monitor='val_acc', min_delta=0, patience=10, verbose=0, mode='auto', baseline=None, restore_best_weights=False)
  history = model.fit(x_train, y_train,
                batch_size=batch_size,
                epochs=epochs,
                validation_data=(x_test, y_test),
                shuffle=True,
                callbacks=[early_stopping])

# Score trained model.
  scores = model.evaluate(x_test, y_test, verbose=1)
  print('Test loss:', scores[0])
  print('Test accuracy:', scores[1])

  return (model, history, scores[0], scores[1])
```

Code Snippet 6

I used Excel to make the following comprehensive table which summarizes all experiments.

ResNet50										
Experiment No.	Epoch	Batch size	Pooling	Optimizer	Early Stop	Cost time	Loss	Acc	Test loss	Test accuracy
1	100	256	None	sgd	22	2574 s	2.5924	0.395	5.056173245	0.0812
2	100	512	None	sgd	40	3880 s	0.8842	0.8115	6.081385613	0.0586
3	200	512	None	sgd	36	3456	1.3857	0.6774	5.8801	0.0589
4	200	256	None	sgd	25	2950	1.884	0.5525	5.402608189	0.0785
5	200	256	None	adam	26	3692	0.6102	0.8474	5.323295047	0.194
6	200	256	avg	sgd	24	2808	2.2329	0.4703	5.14494882	0.0855

Table 2

Through the table, we could find that the model of experiment 5 is the beset model. This model has the highest training accuracy and test accuracy. But the all of the model has the problem of overfitting. Comparing Experiment1,2,3,4, we could find that the increase of batch size would have the bad influence on the result. The reason is the same from above. Comparing Experiment 4,6, we could find that the test accuracy was improved after adding average pooling layer.

5. **Reference**

[1]Tradeoff batch size vs. number of iterations to train a neural network