Part 1 *(code: 351HW2Python_part1.ipynb)*

I split the training set to 80% of training, 20% of validation set, Validation set is the set that want our model to perform well by tuning hyper-parameters. Then, by using Keras library in python, Generate batches of tensor image data with real-time data augmentation for train, validation and test sets. I used the default setting for ImageDataGenerator (will improve this part in part2).

```
Training data
Found 32353 validated image filenames belonging to 13 classes.
Validation data
Found 8088 validated image filenames belonging to 13 classes.
Test data
Found 4000 validated image filenames.
```

Also, by keras library, I generated the following CNN model:

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_21 (Conv2D)           (None, 97, 97, 32)        544
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 48, 48, 32)        0
_________________________________________________________________
dropout_22 (Dropout)         (None, 48, 48, 32)        0
_________________________________________________________________
flatten_6 (Flatten)          (None, 73728)             0
_________________________________________________________________
dense_12 (Dense)             (None, 256)               18874624
_________________________________________________________________
dropout_23 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_13 (Dense)             (None, 13)                3341
=================================================================
Total params: 18,878,509
Trainable params: 18,878,509
Non-trainable params: 0
```

Input layer: reshape image into the input shape that I want. I set the shape=(100, 100, 1)

Conv2d: this layer extract feature from image dataset, has number of fliters that done convolutional operation. I set the kernel size to (4,4), and set activation function to relu, rectified linear unit. It will output the input if input is positive, otherwise it will output 0.

Maxpooling: It reduces volume of input image after convolution 2d.

Flatten: flatten the input, decreases the dimension of input data.

Dense: A layer that receives input from all neurons of its previous layer.

Dropout: Reduce overfitting and complexity of the model.

(I will make improvement for layers in part2.)

So, for the raw model, I choose to use epoch = 2 to lower the computational cost of my computer. Epoch is the number of passes over the training model for the entire dataset.

Here is the result:

```
Epoch 1/2
808/808 [==============================] - 709s 877ms/step - loss: 12.8503 - accuracy: 0.6878 - val_loss: 0.0707 - val_accuracy: 0.8616
Epoch 2/2
808/808 [==============================] - 695s 859ms/step - loss: 0.0789 - accuracy: 0.8529 - val_loss: 0.0612 - val_accuracy: 0.8881
```

```
<tensorflow.python.keras.callbacks.History at 0x1a628d39d88>
```

```python
predict = model.predict_generator(test_generator, steps=num_test_samples //batch_size )

# convert prediction result of integers to categorical names
predicted_class_indices=np.argmax(predict,axis=1)
labels = (training_generator.class_indices)
labels = dict((v,k) for k,v in labels.items())
predictions = [labels[k] for k in predicted_class_indices]
```

```
C:\Users\24937\Anaconda3\lib\site-packages\tensorflow\python\keras\engine\training.py:1905: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, which supports generators.
  warnings.warn('`Model.predict_generator` is deprecated and '
```

```python
from sklearn.metrics import accuracy_score

accuracy_score(test['label'], predictions)
```

```
0.88525
```

By using the test dataset, the result accuracy for the first raw CNN model is 0.88525.

Part 2:

In part 2, according to the previous model, I will make improvement in the following ways:

1. Turning parameters on a dev set. Though I did something in part 1, I can change some parameters, add more layers to the previous model.
2. Generate more image data for training, by keras ImageGenerator.
3. Apply transfer learning using a popular pre-trained image classification.
4. Combine image and text description
5. Hyperparameter pruning (future improvement)

(1) Turning Parameters: According the turning parameters in part1, I change the kernel size from (4,4) to (3,3), change dropout(0.2) to dropout(0.1). For the last dense layer, I choose to use softmax activation function, rather than 'relu'. Softmax converts a real vector to a vector of categorical probabilities.

For the compile method, I choose adam optimization. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. Also, the learning rate is default 0.001.

Also, I add more layers in the model: *(code: 351HW2Python_part2_improved model.ipynb)*

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 97, 97, 32)        544
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 94, 94, 32)        16416
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 47, 47, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 47, 47, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 44, 44, 32)        16416
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 41, 41, 32)        16416
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 20, 20, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 20, 20, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 12800)             0
_________________________________________________________________
dropout_2 (Dropout)          (None, 12800)             0
_________________________________________________________________
dense (Dense)                (None, 256)               3277056
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 13)                3341
=================================================================
Total params: 3,330,189
Trainable params: 3,330,189
Non-trainable params: 0
```

In [12]:

```python
from sklearn.metrics import accuracy_score

accuracy_score(test['label'], predictions)
```

Out[12]: 0.90525

The accuracy is better than part1.

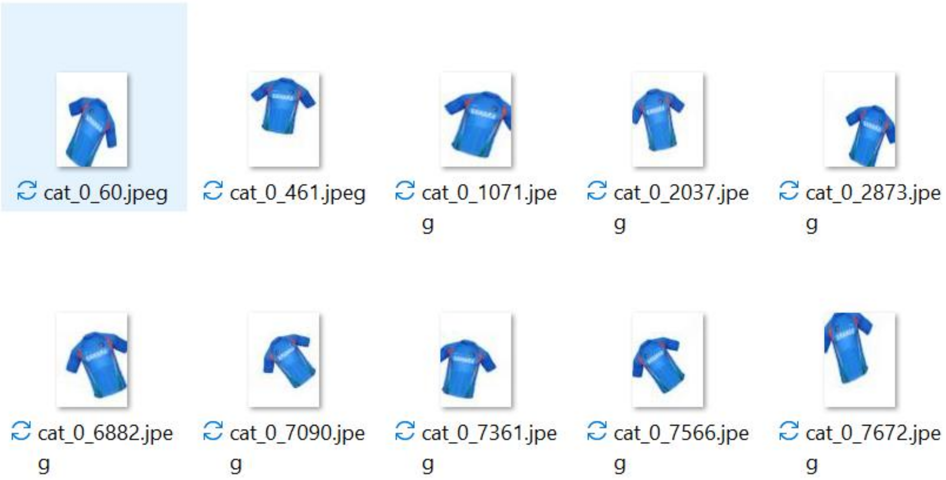
Generate more image data for training: I generated more data by rotation, width & height shift the pictures. As an example of image 1163.jpg:

(2)

Example for (2)

In [29]:

```python
img = load_img('images/1163.jpg')  # this is a PIL image
x = img_to_array(img)  # this is a Numpy array with shape (80, 60, 3)
x = x.reshape((1,) + x.shape)  # this is a Numpy array with shape (1, 80, 60, 3)
i = 0
for batch in image_generator.flow(x, batch_size=1,
                                  save_to_dir='preview', save_prefix='shirt', save_format='jpeg'):
    i += 1
    if i > 20:
        break
```



So, we can create more data for training.

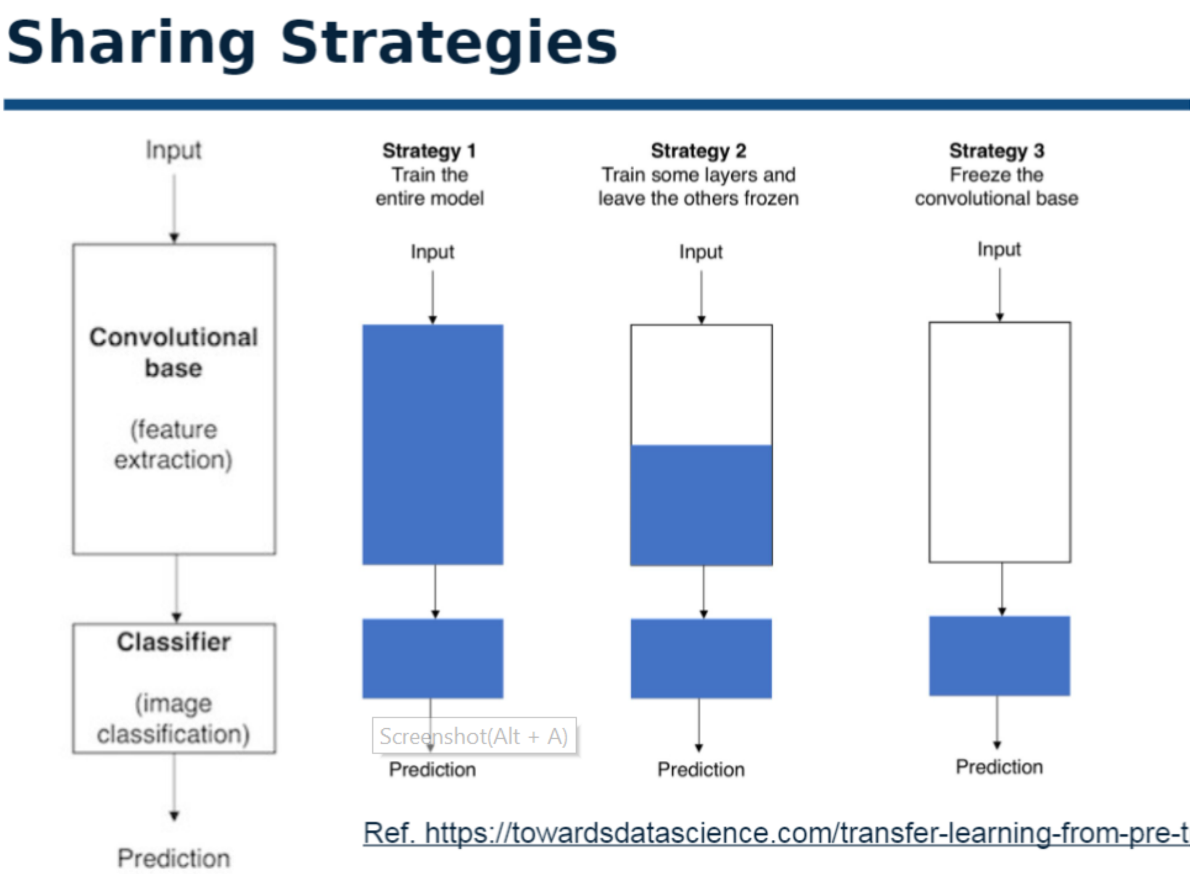
Also I rescale the data to 1/255 and change the activation function to RGB. It is because RGB has coefficients between 0-255.

(3) Apply transfer learning using a popular pre-trained image classification.
I'm trying to do the transfer learning by a pre-trained model. I choose to use Xception because it has the highest accuracy on the website: https://keras.io/api/applications/

(code: 351HW2Python_part2_finalmodel.ipynb)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
xception (Functional)        (None, 3, 3, 2048)        20861480
_________________________________________________________________
global_average_pooling2d_1 ( (None, 2048)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               524544
_________________________________________________________________
dense_2 (Dense)              (None, 13)                3341
=================================================================
Total params: 21,389,365
Trainable params: 527,885
Non-trainable params: 20,861,480
```

It includes the freezed xception layer, global average pooling layer, dropout layer, dense layers.



I'm using strategy 2 in the class. Train the top layer and let other layers frozen, then train the full model.

Then, train the top layer by the above model:

```
# Train the top layer

model.compile(
    optimizer=keras.optimizers.Adam(),
    loss=keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=[keras.metrics.BinaryAccuracy()],
)
epochs = 5
model.fit_generator(
        training_generator,
        steps_per_epoch= 0.8 * num_train_samples // batch_size,
        epochs=epochs, # lower the computational cost
        #verbose=1,
        validation_data=validation_generator,
        validation_steps= 0.2 * num_train_samples // batch_size)
```

```
Epoch 1/5
808/808 [==============================] - 1044s 1s/step - loss: 0.2130 - binary_accuracy: 0.9348 - val_loss: 0.1580 - val_binary_accuracy:
0.9453
Epoch 2/5
808/808 [==============================] - 1075s 1s/step - loss: 0.1550 - binary_accuracy: 0.9447 - val_loss: 0.1427 - val_binary_accuracy:
0.9477
Epoch 3/5
808/808 [==============================] - 1032s 1s/step - loss: 0.1441 - binary_accuracy: 0.9470 - val_loss: 0.1344 - val_binary_accuracy:
0.9481
Epoch 4/5
808/808 [==============================] - 1029s 1s/step - loss: 0.1365 - binary_accuracy: 0.9492 - val_loss: 0.1351 - val_binary_accuracy:
0.9489
Epoch 5/5
808/808 [==============================] - 1067s 1s/step - loss: 0.1330 - binary_accuracy: 0.9501 - val_loss: 0.1274 - val_binary_accuracy:
0.9505
```

At last, do the fine turning for the entire model:

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 100, 100, 3)]     0
_________________________________________________________________
normalization (Normalization (None, 100, 100, 3)       7
_________________________________________________________________
xception (Functional)        (None, 3, 3, 2048)        20861480
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dropout (Dropout)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 256)               524544
_________________________________________________________________
dense_1 (Dense)              (None, 13)                3341
=================================================================
Total params: 21,389,372
Trainable params: 21,334,837
Non-trainable params: 54,535
_________________________________________________________________
808/808 [==============================] - 5051s 6s/step - loss: 0.1684 - binary_accuracy: 0.9420 - val_loss: 0.1128 - val_binary_accuracy: 0.9566
```

I choose a low value of epoch, because of high computational cost (my computer cannot support GPU).

```
from sklearn.metrics import accuracy_score

accuracy_score(test['label'], predictions)

```

```
0.768
```

```
score = model.evaluate(validation_generator)
```

```
203/203 [==============================] - 220s 1s/step - loss: 0.1118 - binary_accuracy: 0.9563
```

The accuracy for testing dataset is low because of low iteration of ‘epoch’ and slow learning rate for the optimization function. It should be higher if epoch = 20 for each step, both training the top layer and the entire model. I choose number of epoch = 5 and 1 in these two steps, because the training time is 20 minutes for each epoch in top layer, and 1 hour for each epoch in the entire model.

The accuracy for validation set is 0.9563.

(4) Combine image and text description
For CNN in text description itself, I created a sample model for it.

(code: 351HW2Python_text analysis.ipynb)

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_6 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_12 (Dense)             (None, 256)               2560256
_________________________________________________________________
dropout_10 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_13 (Dense)             (None, 256)               65792
_________________________________________________________________
dense_14 (Dense)             (None, 13)                3341
=================================================================
Total params: 2,629,389
Trainable params: 2,629,389
Non-trainable params: 0
```

To check if this model works well with description words. This raw model has accuracy 71% by using first 300 descriptions in training dataset and all descriptions in testing dataset.

Then, combine the model with description words with the transformation model, to see the performance increases or not. (I choose epoch = 1 because it runs for 1 hour and 20 minutes)

(code: 351HW2Python_part2_final model.ipynb)

```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, 100, 100, 3) 0
__________________________________________________________________________________________________
normalization (Normalization)   (None, 100, 100, 3)  7           input_2[0][0]
__________________________________________________________________________________________________
flatten_1_input (InputLayer)    [(None, 10000)]      0
__________________________________________________________________________________________________
xception (Functional)           (None, 3, 3, 2048)   20861480    normalization[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 10000)        0           flatten_1_input[0][0]
__________________________________________________________________________________________________
global_average_pooling2d (Globa (None, 2048)         0           xception[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 256)          2560256     flatten_1[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 2048)         0           global_average_pooling2d[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 256)          0           dense_5[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 256)          524544      dropout[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 256)          65792       dropout_2[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 13)           3341        dense[0][0]
__________________________________________________________________________________________________
dense_7 (Dense)                 (None, 13)           3341        dense_6[0][0]
__________________________________________________________________________________________________
add_13 (Add)                    (None, 13)           0           dense_1[0][0]
                                                                 dense_7[0][0]
==================================================================================================
Total params: 24,018,761
Trainable params: 23,964,226
Non-trainable params: 54,535
```

The solution is:

```
1138/1138 [==============================] - 5103s 4s/step - loss: 120.4788 - binary_accuracy: 0.9074 - val_loss: 0.1687 - val_binary_accuracy: 0.9231
```

```
score = newModel.evaluate([X_test_p,x_test], y_test, batch_size=batch_size, verbose=1)
```

```
125/125 [==============================] - 114s 838ms/step - loss: 0.1701 - binary_accuracy: 0.9231
```

The validation binary accuracy is 0.9231, the binary accuracy for test set is 0.9231. It is better than the previous model, CNN with transform learning.

(5) Hyperparameter pruning (further improvement)
By keras turner API, we can set different parameters such as learning rate, loss function, to get the optimal answer for our current model.

Summary:

I created a basic CNN model in part 1. Then in part 2, I improve the previous CNN model in part 1, then using transfer learning with Xception pre-trained model. After that, I combined text analysis with descriptions of clothes, made the performance of model better. Also, Hyper parameter pruning can optimize the current model performance.

The further improvement could be, run the model by GPU, that can allow my computer to have more iteration, or larger number of epochs. For most of the models I run the parameter for epoch between 1-5, which is a low number of epoch.