HW3 Pattern Recognition

2019150445 신백록

1. Download MNIST dataset

In [1]:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

(X,y),(X_test,y_test)=mnist.load_data()
print(X.shape)
print(X_test.shape)
print(y.shape)
print(y_test.shape)
```

```
Init Plugin
Init Graph Optimizer
Init Kernel
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
```

In [2]:

```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1, stra
```

In [3]:

```
print(X_train.shape)
print(X_val.shape)
```

```
(54000, 28, 28)
(6000, 28, 28)
```

By train_test_split function, I assigned 10% of training set to X_val, y_val.

In [4]:

```
X_train=X_train.reshape(-1,28*28)
X_val=X_val.reshape(-1,28*28)
X_test=X_test.reshape(-1,28*28)
print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```

```
(54000, 784)
(6000, 784)
(10000, 784)
```

In [5]:

```
from tensorflow.keras.utils import to_categorical

X_train=X_train/255.
X_val=X_val/255.
X_test=X_test/255.

y_train=to_categorical(y_train)
y_val=to_categorical(y_val)
y_test=to_categorical(y_test)
print(y_train.shape)
```

```
(54000, 10)
```

I reshaped the image data to flatten data to use MLP. And by to_categorical function, I made y to be one-hot encoded.

2. Explain Activation Function

The activation function makes the linear combination $w^T x$ to $g(w^T x)$ to make the linear function non-linear. Since linear functions cannot create a non-linear model, activation functions are used. In fact, if each layer is connected only by a linear combination without an activation function, the result will be the same as using a single linear combination even if several hidden layers are created.

For example, the sigmoid function transforms the result of a linear combination non-linearly so that it falls within the range 0 to 1. Therefore, it is often used for the last MLP layer in a binary classifcation model that classifies 0 or 1.

If the result of the linear combination is less than 0, the ReLU function converts it to 0, and if it is greater than 0, the result is converted as it is. Since the differential value is 0 or 1, gradient vanishing and exploding phenomena that occur when the layer is deep can be prevented. In addition, since the differential calculation is simple, it has the advantage of fast learning speed.

In conclusion, activation functions such as sigmoid and ReLU functions are used to create non-linear models that cannot be expressed as linear combinations.

3. Explain MLP

Let's define input value of unit j as x_j, weight on link from unit j to unit i as θ_{ij}, activation function as $g(.)$, activation value of unit i as a_i

Assume there are 1 input layer and 1 hidden layer with 3 nodes each excluding bias node. Then nodes of input layer is x_0(actually it is bias term 1), x_1, x_2 and x_3. The node for next layer a_i is defined with linear combination of input nodes and activation function. For example, $a_1 = g(\theta_{10}x_0 + \theta_{11}x_1 + \theta_{12}x_2 + \theta_{13}x_3)$ and $a_2 = g(\theta_{20}x_0 + \theta_{21}x_1 + \theta_{22}x_2 + \theta_{23}x_3)$

Here, our purpose is to find optimal parameter θ_{ij}. So first, we initialize θ_{ij} to any value or something. Then we compute error at last output layer for initialized parameter θ. And compute error for each node at each layer and by optimization algorithm(gradient descent algorithm or something), move θ_{ij} to reduce the error. Then by that θ we obtain, we again do forward propagation and compute error and back propagation again and again until obtain optimal parameter.

4. Training and Evaluation

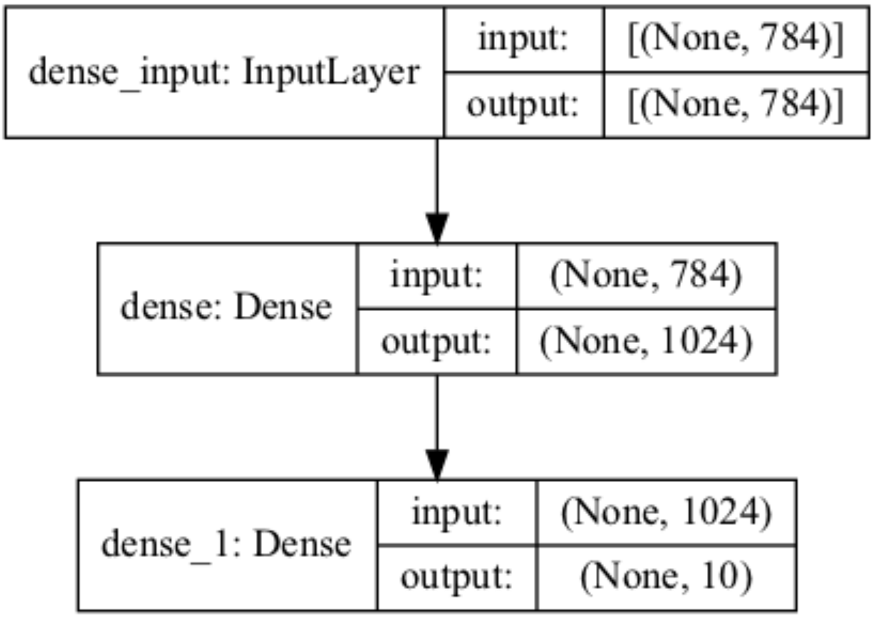
In [6]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import plot_model
```

In [7]:
```
model=Sequential()
model.add(Dense(1024, input_dim=784, activation='relu'))
model.add(Dense(10, activation='relu'))
model.summary()

plot_model(model,show_shapes=True)
```

```
Metal device set to: Apple M1
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1024)              803840
_________________________________________________________________
dense_1 (Dense)              (None, 10)                10250
=================================================================
Total params: 814,090
Trainable params: 814,090
Non-trainable params: 0
_________________________________________________________________
2021-12-01 16:42:44.159086: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA support.
2021-12-01 16:42:44.159170: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id: <undefined>)
```

Out[7]:



In [8]:

```python
import keras
test_acc=[]
class TestCallback(keras.callbacks.Callback):
    def __init__(self, test_data):
        self.test_data = test_data

    def on_epoch_end(self, epoch, logs={}):
        global test_acc
        x, y = self.test_data
        test_acc.append(self.model.evaluate(x, y, verbose=0)[1])
```

By callback function, test accuracy for each epoch is calculated and saved in test_acc

In [9]:
```
model.compile(loss='categorical_crossentropy',optimizer='adam', metrics=['a
```

In [10]:
```
history=model.fit(X_train, y_train, epochs=20, batch_size=64, validation_da
history_out=history.history
```

```
Epoch 1/20
  1/844 [..............................] - ETA: 2:23 - loss: 10.1765 - accu
racy: 0.0938
2021-12-01 16:42:46.159026: I tensorflow/compiler/mlir/mlir_graph_optimizat
ion_pass.cc:176] None of the MLIR Optimization Passes are enabled (register
ed 2)
2021-12-01 16:42:46.159177: W tensorflow/core/platform/profile_utils/cpu_ut
ils.cc:128] Failed to get CPU frequency: 0 Hz
2021-12-01 16:42:46.253955: I tensorflow/core/grappler/optimizers/custom_gr
aph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enab
led.
844/844 [==============================] - ETA: 0s - loss: 4.9966 - accurac
y: 0.6053
2021-12-01 16:42:52.341683: I tensorflow/core/grappler/optimizers/custom_gr
aph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enab
led.
844/844 [==============================] - 6s 7ms/step - loss: 4.9966 - acc
uracy: 0.6053 - val_loss: 5.2325 - val_accuracy: 0.4743
Epoch 2/20
844/844 [==============================] - 6s 7ms/step - loss: 4.9795 - acc
uracy: 0.6016 - val_loss: 5.0079 - val_accuracy: 0.6152
Epoch 3/20
844/844 [==============================] - 6s 7ms/step - loss: 4.8949 - acc
uracy: 0.6384 - val_loss: 4.8330 - val_accuracy: 0.6617
Epoch 4/20
844/844 [==============================] - 6s 7ms/step - loss: 4.7865 - acc
uracy: 0.6187 - val_loss: 5.1388 - val_accuracy: 0.5480
Epoch 5/20
844/844 [==============================] - 6s 7ms/step - loss: 4.6125 - acc
uracy: 0.6511 - val_loss: 4.8430 - val_accuracy: 0.6723
Epoch 6/20
844/844 [==============================] - 6s 7ms/step - loss: 4.8337 - acc
uracy: 0.6263 - val_loss: 4.8255 - val_accuracy: 0.6503
Epoch 7/20
844/844 [==============================] - 6s 7ms/step - loss: 4.7200 - acc
uracy: 0.6364 - val_loss: 4.8310 - val_accuracy: 0.6683
Epoch 8/20
844/844 [==============================] - 6s 7ms/step - loss: 4.3247 - acc
uracy: 0.6621 - val_loss: 4.4427 - val_accuracy: 0.6638
Epoch 9/20
844/844 [==============================] - 6s 7ms/step - loss: 4.2171 - acc
uracy: 0.6610 - val_loss: 4.7152 - val_accuracy: 0.6567
Epoch 10/20
844/844 [==============================] - 6s 7ms/step - loss: 4.7627 - acc
uracy: 0.6477 - val_loss: 4.7909 - val_accuracy: 0.6762
Epoch 11/20
844/844 [==============================] - 6s 7ms/step - loss: 4.6253 - acc
uracy: 0.6743 - val_loss: 4.5289 - val_accuracy: 0.6785
```

```
Epoch 12/20
844/844 [==============================] - 6s 7ms/step - loss: 4.3116 - acc
uracy: 0.6746 - val_loss: 4.0436 - val_accuracy: 0.6783
Epoch 13/20
844/844 [==============================] - 6s 7ms/step - loss: 4.1727 - acc
uracy: 0.6751 - val_loss: 4.8201 - val_accuracy: 0.6423
Epoch 14/20
844/844 [==============================] - 6s 7ms/step - loss: 4.4683 - acc
uracy: 0.6688 - val_loss: 4.5122 - val_accuracy: 0.6700
Epoch 15/20
844/844 [==============================] - 6s 7ms/step - loss: 3.6945 - acc
uracy: 0.6708 - val_loss: 3.4961 - val_accuracy: 0.6767
Epoch 16/20
844/844 [==============================] - 6s 7ms/step - loss: 2.8948 - acc
uracy: 0.6666 - val_loss: 2.5915 - val_accuracy: 0.6598
Epoch 17/20
844/844 [==============================] - 6s 7ms/step - loss: 3.6029 - acc
uracy: 0.6678 - val_loss: 4.0397 - val_accuracy: 0.6735
Epoch 18/20
844/844 [==============================] - 6s 7ms/step - loss: 3.8680 - acc
uracy: 0.6784 - val_loss: 3.7478 - val_accuracy: 0.6808
Epoch 19/20
844/844 [==============================] - 6s 7ms/step - loss: 3.5250 - acc
uracy: 0.6757 - val_loss: 3.6131 - val_accuracy: 0.6765
Epoch 20/20
844/844 [==============================] - 6s 7ms/step - loss: 3.7291 - acc
uracy: 0.6745 - val_loss: 3.6157 - val_accuracy: 0.6813
```

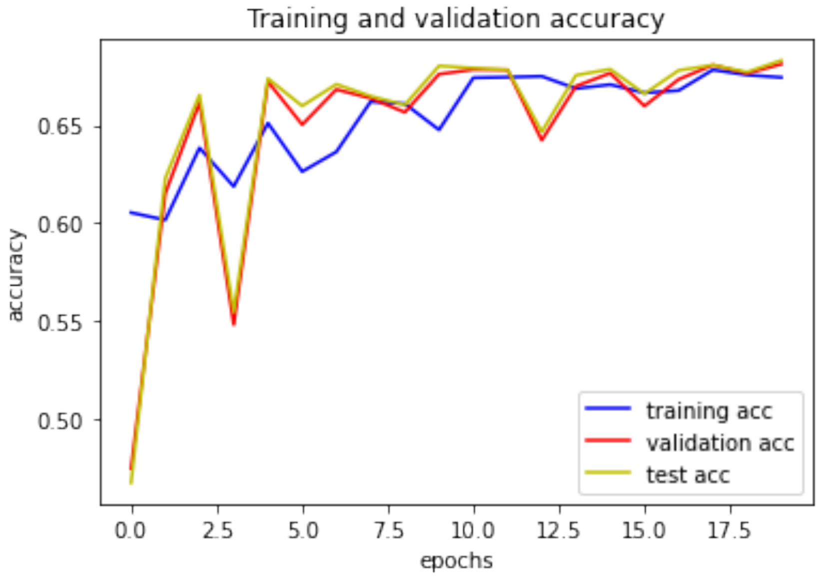
In [11]:

```
import matplotlib.pyplot as plt

accuracy=history_out['accuracy']
accuracy_val=history_out['val_accuracy']

plt.clf()
plt.plot(accuracy,'b',label='training acc')
plt.plot(accuracy_val,'b', color='r', label='validation acc' )
plt.plot(test_acc, 'b', color='y', label='test acc')
plt.title('Training and validation accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```

```
/var/folders/9r/7f1bw3q15yz7435178lsgb5w0000gp/T/ipykernel_3430/2521518520.
py:8: UserWarning: color is redundantly defined by the 'color' keyword argu
ment and the fmt string "b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword argu
ment will take precedence.
  plt.plot(accuracy_val,'b', color='r', label='validation acc' )
/var/folders/9r/7f1bw3q15yz7435178lsgb5w0000gp/T/ipykernel_3430/2521518520.
py:9: UserWarning: color is redundantly defined by the 'color' keyword argu
ment and the fmt string "b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword argu
ment will take precedence.
  plt.plot(test_acc, 'b', color='y', label='test acc')
```



We can see there are no convergences occur for train, validation, test set for first 12 epochs. But after 12 epoch, we can see some convergence for train, validation, test data set. But accuracy is too low which is underfitted.

In [13]:

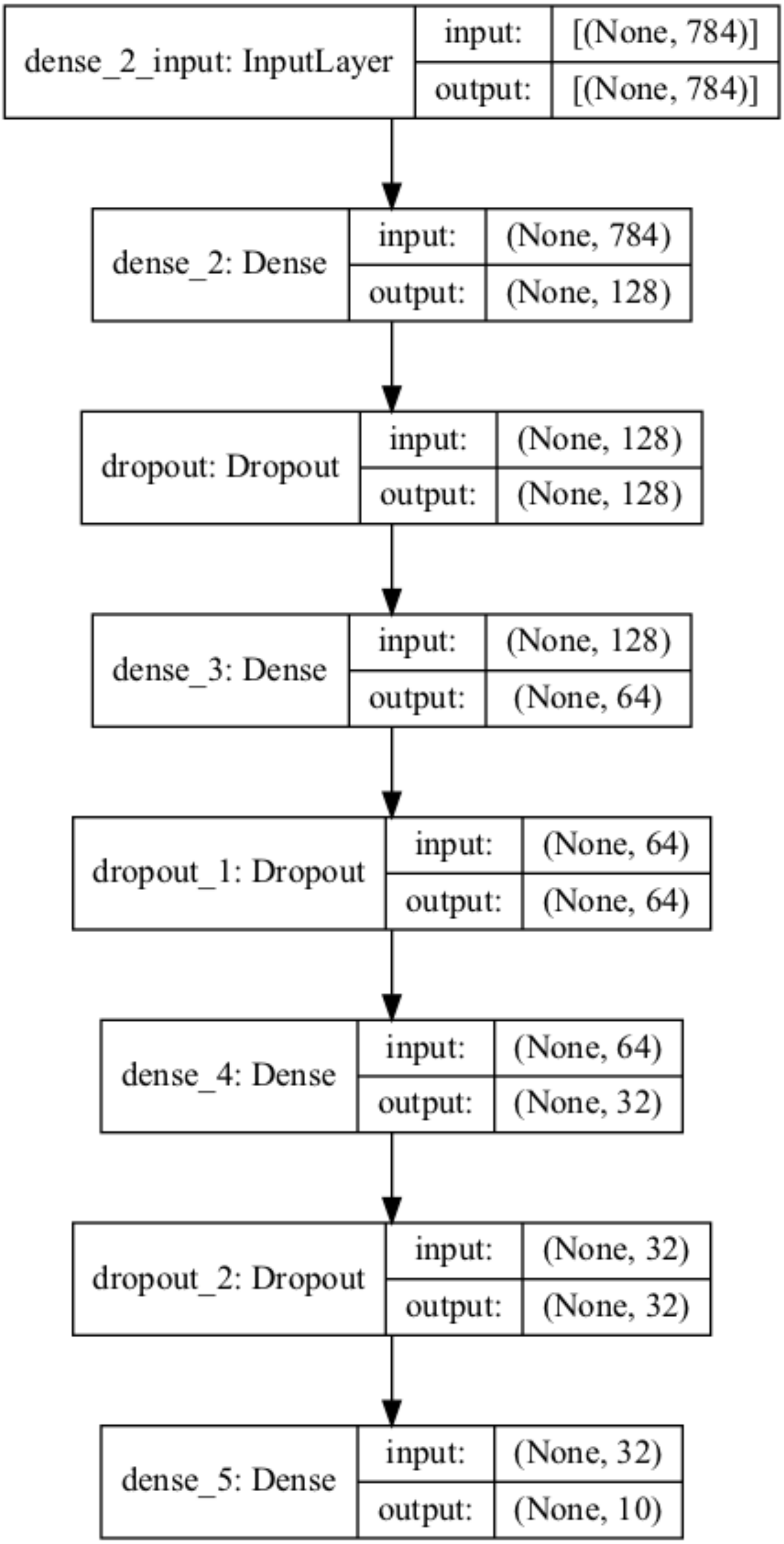
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout, BatchNormal
from tensorflow.keras.utils import plot_model

model=Sequential()
model.add(Dense(128, input_dim=784, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10,activation='softmax'))
model.summary()
plot_model(model,show_shapes=True)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 128)               100480
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
```

```
dense_3 (Dense)              (None, 64)                8256      
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080      
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_5 (Dense)              (None, 10)                330       
=================================================================
Total params: 111,146
Trainable params: 111,146
Non-trainable params: 0
_________________________________________________________________
```

Out[13]:


dense_2_input: InputLayer
input: [(None, 784)]
output: [(None, 784)]
dense_2: Dense
input: (None, 784)
output: (None, 128)
dropout: Dropout
input: (None, 128)
output: (None, 128)
dense_3: Dense
input: (None, 128)
output: (None, 64)
dropout_1: Dropout
input: (None, 64)
output: (None, 64)
dense_4: Dense
input: (None, 64)
output: (None, 32)
dropout_2: Dropout
input: (None, 32)
output: (None, 32)
dense_5: Dense
input: (None, 32)
output: (None, 10)

Input dim is set to 784 because there are 784 characteristic variables.

The first Dense layer was fitted with dimension 128 and the activation function was relu.

Dropout was given to prevent overfitting, so that each node had a value of 0 with a 40% probability.

The second Dense layer is set to dimension 64, and the activation function is relu as above, and gave a dropout of 0.3.

The third Dense layer is set to dimension 32, activation function relu, and dropout 0.2.

The last Dense layer has to extract the probability for 10 labels, so dimension=10, activation='softmax' was given.

In order to prevent the bottleneck phenomenon, the dimension gradually decreases from 128, and finally at the last layer, 10 nodes are estimated from 32 nodes.

In [14]:

```
test_acc=[]
#initialize test_acc list.
```

In [15]:

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
callback_list=[TestCallback((X_test, y_test)),EarlyStopping(monitor='val_ac
```

Earlystopping and model checkpoint were called to give callback, and to prevent overfitting, learning was stopped if validation_accuracy did not improve for 4 epochs, and the model was saved when val_accuracy was improved.

In [16]:

```
model.compile(loss='categorical_crossentropy',optimizer='adam', metrics=['a
history=model.fit(X_train, y_train, epochs=30, batch_size=64, callbacks=ca
history_out=history.history
history_out.keys()
```

```
Epoch 1/30
 16/591 [..............................] - ETA: 4s - loss: 2.2107 - accurac
y: 0.1660
2021-12-01 16:49:06.047873: I tensorflow/core/grappler/optimizers/custom_gr
aph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enab
led.
591/591 [==============================] - ETA: 0s - loss: 0.7558 - accurac
y: 0.7616
2021-12-01 16:49:09.952388: I tensorflow/core/grappler/optimizers/custom_gr
aph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enab
led.
591/591 [==============================] - 5s 8ms/step - loss: 0.7558 - acc
uracy: 0.7616 - val_loss: 0.2443 - val_accuracy: 0.9298
Epoch 2/30
591/591 [==============================] - 5s 8ms/step - loss: 0.3655 - acc
uracy: 0.9007 - val_loss: 0.1977 - val_accuracy: 0.9435
```

```
Epoch 3/30
591/591 [==============================] - 5s 8ms/step - loss: 0.2877 - acc
uracy: 0.9216 - val_loss: 0.1555 - val_accuracy: 0.9557
Epoch 4/30
591/591 [==============================] - 5s 8ms/step - loss: 0.2489 - acc
uracy: 0.9308 - val_loss: 0.1460 - val_accuracy: 0.9578
Epoch 5/30
591/591 [==============================] - 5s 8ms/step - loss: 0.2216 - acc
uracy: 0.9391 - val_loss: 0.1252 - val_accuracy: 0.9641
Epoch 6/30
591/591 [==============================] - 5s 8ms/step - loss: 0.2003 - acc
uracy: 0.9463 - val_loss: 0.1308 - val_accuracy: 0.9622
Epoch 7/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1868 - acc
uracy: 0.9484 - val_loss: 0.1158 - val_accuracy: 0.9669
Epoch 8/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1798 - acc
uracy: 0.9508 - val_loss: 0.1171 - val_accuracy: 0.9678
Epoch 9/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1649 - acc
uracy: 0.9539 - val_loss: 0.1161 - val_accuracy: 0.9682
Epoch 10/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1578 - acc
uracy: 0.9562 - val_loss: 0.1071 - val_accuracy: 0.9700
Epoch 11/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1533 - acc
uracy: 0.9568 - val_loss: 0.1053 - val_accuracy: 0.9701
Epoch 12/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1487 - acc
uracy: 0.9592 - val_loss: 0.1041 - val_accuracy: 0.9704
Epoch 13/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1405 - acc
uracy: 0.9606 - val_loss: 0.1038 - val_accuracy: 0.9717
Epoch 14/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1364 - acc
uracy: 0.9621 - val_loss: 0.1015 - val_accuracy: 0.9726
Epoch 15/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1283 - acc
uracy: 0.9639 - val_loss: 0.1012 - val_accuracy: 0.9715
Epoch 16/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1288 - acc
uracy: 0.9637 - val_loss: 0.1022 - val_accuracy: 0.9727
Epoch 17/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1202 - acc
uracy: 0.9651 - val_loss: 0.0953 - val_accuracy: 0.9741
Epoch 18/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1202 - acc
uracy: 0.9653 - val_loss: 0.1007 - val_accuracy: 0.9735
Epoch 19/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1165 - acc
uracy: 0.9669 - val_loss: 0.0989 - val_accuracy: 0.9738
Epoch 20/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1124 - acc
uracy: 0.9670 - val_loss: 0.0929 - val_accuracy: 0.9752
Epoch 21/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1114 - acc
uracy: 0.9681 - val_loss: 0.0938 - val_accuracy: 0.9754
```

```
Epoch 22/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1095 - acc
uracy: 0.9682 - val_loss: 0.0946 - val_accuracy: 0.9743
Epoch 23/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1054 - acc
uracy: 0.9697 - val_loss: 0.0973 - val_accuracy: 0.9740
Epoch 24/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1027 - acc
uracy: 0.9700 - val_loss: 0.1036 - val_accuracy: 0.9741
Epoch 25/30
591/591 [==============================] - 5s 8ms/step - loss: 0.1041 - acc
uracy: 0.9696 - val_loss: 0.1004 - val_accuracy: 0.9736
```

Out[16]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

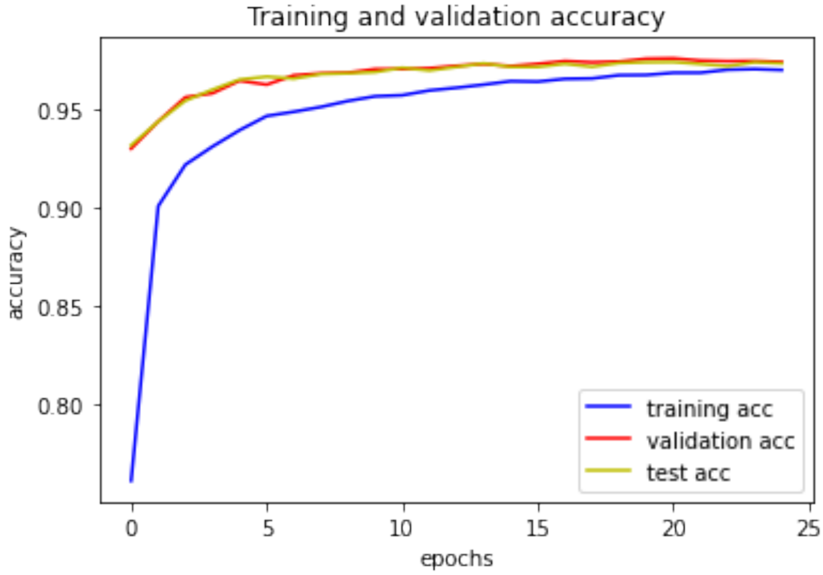
In [17]:

```python
import matplotlib.pyplot as plt

accuracy=history_out['accuracy']
accuracy_val=history_out['val_accuracy']

plt.clf()
plt.plot(accuracy,'b',label='training acc')
plt.plot(accuracy_val,'b', color='r', label='validation acc' )
plt.plot(test_acc, 'b', color='y', label='test acc')
plt.title('Training and validation accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```

```
/var/folders/9r/7f1bw3q15yz7435178lsgb5w0000gp/T/ipykernel_3430/2521518520.
py:8: UserWarning: color is redundantly defined by the 'color' keyword argu
ment and the fmt string "b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword argu
ment will take precedence.
  plt.plot(accuracy_val,'b', color='r', label='validation acc' )
/var/folders/9r/7f1bw3q15yz7435178lsgb5w0000gp/T/ipykernel_3430/2521518520.
py:9: UserWarning: color is redundantly defined by the 'color' keyword argu
ment and the fmt string "b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword argu
ment will take precedence.
  plt.plot(test_acc, 'b', color='y', label='test acc')
```



It converges neatly without any vibration.