Mehrdad Baradaran

# Convolutional neural network

Cifar_10 Dtaset

Assignment 2

# Headline:

- depth effect of different hidden layers.
- batch normalization for convolution layers
- different architectures to find the optimum model
- confusion matrix

At first, we create our convolutional model and with 3 hidden layers, we try to extract deeper details from the image with convolution, and finally we flatten the data so that we can predict the content of the image using softmax, here to avoid train Too much model and overfit, we use the early stopping method, give the train data to the model and start modeling, the result will be as follows.

Before examining the details of train, we should know that in order to classify the labels, we display the output of the labels as one-hot, in which the content index of the photo becomes 1, and the rest is 0. As a result, if the photo of a cat Let's give E to the model and assume that the cat is our class number 3, the output will be [0, 0, 0, 1, 0, 0, 0, 0, 0, 0,].

## import to_categorical method for converting labels to one-hot

In converting the labels to one-hot, instead of displaying the number of each class, we create an array with the length of the number of classes, and in that we display the probability of the existence of each class, which is used in such a way that all the elements of the array are zero and the index to which the photo belongs that class is one. Or is it that all the numerical elements are between 0 or 1, that the probability of each class in that photo is shown . the sum of all the numbers must be equal to 1.

In [11]:
```
from tensorflow.keras.utils import to_categorical
```

In [12]:
```
train_cat_y = to_categorical(train_y, num_classes=10)
train_cat_y.shape, train_cat_y.dtype
```

Out[12]:
```
((50000, 10), dtype('float32'))
```

In [13]:
```
test_cat_y = to_categorical(test_y, num_classes=10)
test_cat_y.shape, test_cat_y.dtype
```

Out[13]:
```
((10000, 10), dtype('float32'))
```

In [25]:

```
input = Input(shape=(32, 32, 3))

#block1
x = Conv2D(filters=32, kernel_size=(5, 5), strides=1, padding='same')(input)
x = ReLU()(x)
x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

#block2
x = Conv2D(filters=64, kernel_size=(3, 3), strides=1, padding='same')(x)
x = ReLU()(x)
x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

#block3
x = Conv2D(filters=128, kernel_size=(3, 3), strides=1, padding='same')(x)
x = ReLU()(x)
x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

#flatten
x = Flatten()(x)

# fully connected
x = Dense(units=128)(x)
x = ReLU()(x)

x = Dense(units=10)(x)
output = Activation(activation='softmax')(x)

cnn_model = Model(input, output)
cnn_model.summary()
```

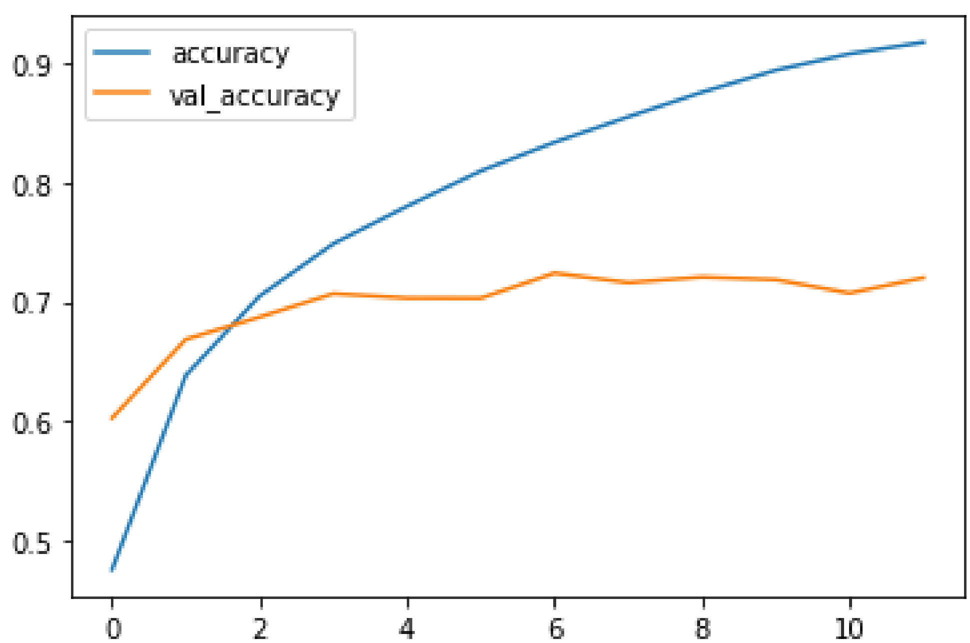
Summary of our Model :

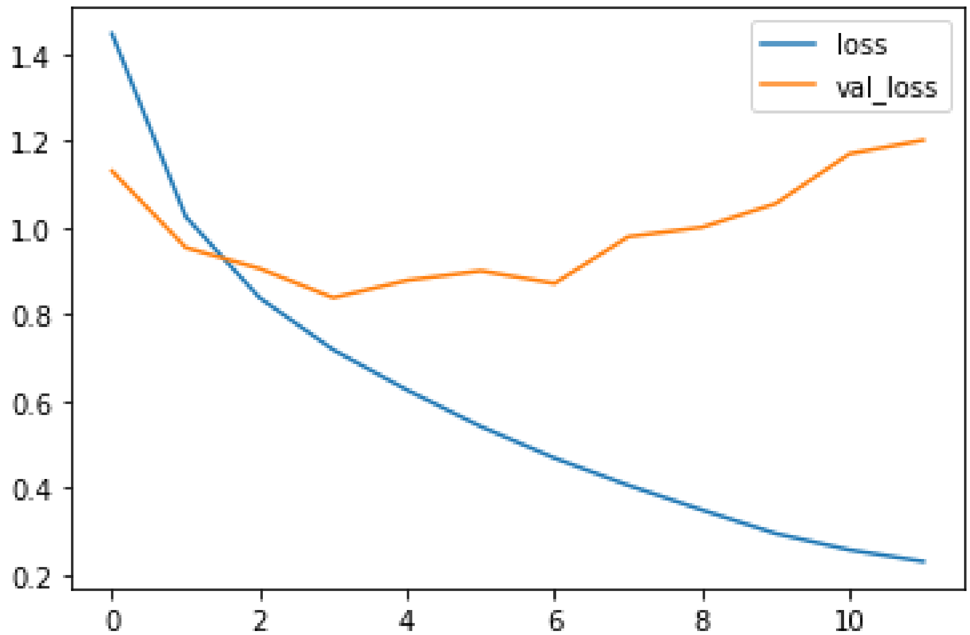
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 32, 32, 3)]       0         
                                                                 
 conv2d_3 (Conv2D)           (None, 32, 32, 32)        2432      
                                                                 
 re_lu_4 (ReLU)              (None, 32, 32, 32)        0         
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 16, 16, 32)       0         
 2D)                                                             
                                                                 
 conv2d_4 (Conv2D)           (None, 16, 16, 64)        18496     
                                                                 
 re_lu_5 (ReLU)              (None, 16, 16, 64)        0         
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 8, 8, 64)         0         
 2D)                                                             
                                                                 
 conv2d_5 (Conv2D)           (None, 8, 8, 128)         73856     
                                                                 
 re_lu_6 (ReLU)              (None, 8, 8, 128)         0         
                                                                 
 max_pooling2d_5 (MaxPooling  (None, 4, 4, 128)        0         
 2D)                                                             
                                                                 
 flatten_1 (Flatten)         (None, 2048)              0         
                                                                 
 dense_1 (Dense)             (None, 128)               262272    
                                                                 
 re_lu_7 (ReLU)              (None, 128)               0         
                                                                 
 dense_2 (Dense)             (None, 10)                1290      
                                                                 
 activation (Activation)     (None, 10)                0         
                                                                 
=================================================================
Total params: 358,346
Trainable params: 358,346
Non-trainable params: 0
_________________________________________
```

Our Model has 358346 parameters .

We build the Early stopping method based on val_accuracy to stop the train if its value does not increase by 5 epochs.

```
Epoch 1/30
1563/1563 [==============================] - 100s 63ms/step - loss: 1.4461 - accuracy
: 0.4758 - val_loss: 1.1300 - val_accuracy: 0.6029
Epoch 2/30
1563/1563 [==============================] - 104s 66ms/step - loss: 1.0247 - accuracy
: 0.6387 - val_loss: 0.9538 - val_accuracy: 0.6689
Epoch 3/30
1563/1563 [==============================] - 102s 65ms/step - loss: 0.8383 - accuracy
: 0.7050 - val_loss: 0.9057 - val_accuracy: 0.6874
Epoch 4/30
1563/1563 [==============================] - 110s 70ms/step - loss: 0.7182 - accuracy
: 0.7489 - val_loss: 0.8383 - val_accuracy: 0.7071
Epoch 5/30
1563/1563 [==============================] - 103s 66ms/step - loss: 0.6254 - accuracy
: 0.7804 - val_loss: 0.8783 - val_accuracy: 0.7034
Epoch 6/30
1563/1563 [==============================] - 104s 67ms/step - loss: 0.5417 - accuracy
: 0.8099 - val_loss: 0.9002 - val_accuracy: 0.7033
Epoch 7/30
1563/1563 [==============================] - 103s 66ms/step - loss: 0.4689 - accuracy
: 0.8340 - val_loss: 0.8714 - val_accuracy: 0.7243
Epoch 8/30
1563/1563 [==============================] - 105s 67ms/step - loss: 0.4061 - accuracy
: 0.8553 - val_loss: 0.9787 - val_accuracy: 0.7163
Epoch 9/30
1563/1563 [==============================] - 105s 67ms/step - loss: 0.3492 - accuracy
: 0.8760 - val_loss: 1.0004 - val_accuracy: 0.7210
Epoch 10/30
1563/1563 [==============================] - 104s 66ms/step - loss: 0.2949 - accuracy
: 0.8944 - val_loss: 1.0551 - val_accuracy: 0.7188
Epoch 11/30
1563/1563 [==============================] - 105s 67ms/step - loss: 0.2571 - accuracy
: 0.9082 - val_loss: 1.1698 - val_accuracy: 0.7075
Epoch 12/30
1563/1563 [==============================] - 103s 66ms/step - loss: 0.2306 - accuracy
: 0.9178 - val_loss: 1.2014 - val_accuracy: 0.7204
```






Accuracy was increasing, but val_accuracy remains constant, but in the next graph, val_loss starts to increase after 9 epochs, and this means that our model is increasing too much compared to the train data, so we used the early stop function to When the model is finished at the time when it is in its best state and we store all the weights in that epoch to use the model in its optimal state.

After evaluating model on test data :

```
313/313 [==============================] - 5s 16ms/step - loss: 0.8714 - accuracy: 0.
7243
```

We reach an acceptable accuracy because the data were heavy photos in RGB format.

```python
In [51]: input = Input(shape=(32, 32, 3))

#block1
x = Conv2D(filters=32, kernel_size=(7, 7), strides=1, padding='same')(input)
x = ReLU()(x)
x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

#block2
x = Conv2D(filters=64, kernel_size=(5, 5), strides=1, padding='same')(x)
x = ReLU()(x)
x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

#block3
x = Conv2D(filters=128, kernel_size=(3, 3), strides=1, padding='same')(x)
x = ReLU()(x)
x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

#block4
x = Conv2D(filters=256, kernel_size=(3, 3), strides=1, padding='same')(x)
x = ReLU()(x)
x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

#globalaveragepooling
x = GlobalAveragePooling2D()(x)

# fully connected
x = Dense(units=128)(x)
x = ReLU()(x)
x = Dense(units=32)(x)
x = ReLU()(x)
x = Dense(units=10)(x)
output = Activation(activation='softmax')(x)

cnn_model2 = Model(input, output)
cnn_model2.summary()
```

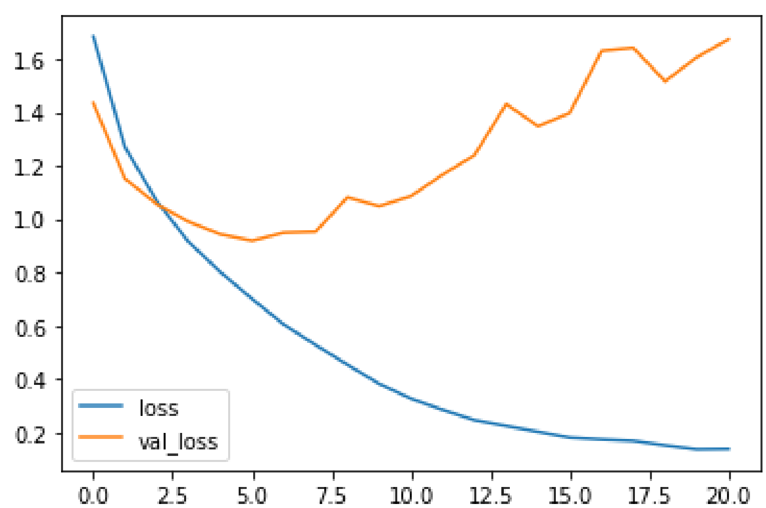
Next time, by adding a layer to the model and using global average pooling, we tried to reduce the parameters by increasing the hidden layers. It was expected that the train model would be better.

But :

```
Epoch 1/30
1563/1563 [==============================] - 212s 135ms/step - loss: 1.6838 - accurac
y: 0.3710 - val_loss: 1.4338 - val_accuracy: 0.4726
Epoch 2/30
1563/1563 [==============================] - 220s 141ms/step - loss: 1.2687 - accurac
y: 0.5411 - val_loss: 1.1507 - val_accuracy: 0.5866
Epoch 3/30
1563/1563 [==============================] - 224s 143ms/step - loss: 1.0663 - accurac
y: 0.6192 - val_loss: 1.0549 - val_accuracy: 0.6240
Epoch 4/30
1563/1563 [==============================] - 221s 141ms/step - loss: 0.9139 - accurac
y: 0.6793 - val_loss: 0.9896 - val_accuracy: 0.6522
Epoch 5/30
1563/1563 [==============================] - 220s 141ms/step - loss: 0.8010 - accurac
y: 0.7190 - val_loss: 0.9426 - val_accuracy: 0.6752
Epoch 6/30
1563/1563 [==============================] - 215s 138ms/step - loss: 0.7001 - accurac
y: 0.7547 - val_loss: 0.9185 - val_accuracy: 0.6875
Epoch 7/30
1563/1563 [==============================] - 215s 138ms/step - loss: 0.6039 - accurac
y: 0.7887 - val_loss: 0.9488 - val_accuracy: 0.6858
Epoch 8/30
1563/1563 [==============================] - 217s 139ms/step - loss: 0.5275 - accurac
y: 0.8157 - val_loss: 0.9523 - val_accuracy: 0.6937
```

```
Epoch 9/30
1563/1563 [==============================] - 220s 141ms/step - loss: 0.4535 - accurac
y: 0.8399 - val_loss: 1.0808 - val_accuracy: 0.6839
Epoch 10/30
1563/1563 [==============================] - 219s 140ms/step - loss: 0.3822 - accurac
y: 0.8673 - val_loss: 1.0478 - val_accuracy: 0.6890
Epoch 11/30
1563/1563 [==============================] - 215s 138ms/step - loss: 0.3265 - accurac
y: 0.8865 - val_loss: 1.0853 - val_accuracy: 0.7016
Epoch 12/30
1563/1563 [==============================] - 217s 139ms/step - loss: 0.2846 - accurac
y: 0.9013 - val_loss: 1.1662 - val_accuracy: 0.6967
Epoch 13/30
1563/1563 [==============================] - 216s 138ms/step - loss: 0.2460 - accurac
y: 0.9138 - val_loss: 1.2388 - val_accuracy: 0.6954
Epoch 14/30
1563/1563 [==============================] - 216s 138ms/step - loss: 0.2242 - accurac
y: 0.9205 - val_loss: 1.4305 - val_accuracy: 0.6703
Epoch 15/30
1563/1563 [==============================] - 215s 138ms/step - loss: 0.2023 - accurac
y: 0.9284 - val_loss: 1.3471 - val_accuracy: 0.6910
Epoch 16/30
1563/1563 [==============================] - 218s 139ms/step - loss: 0.1806 - accurac
y: 0.9378 - val_loss: 1.3969 - val_accuracy: 0.6887
Epoch 17/30
1563/1563 [==============================] - 219s 140ms/step - loss: 0.1737 - accurac
y: 0.9402 - val_loss: 1.6299 - val_accuracy: 0.6842
Epoch 18/30
1563/1563 [==============================] - 211s 135ms/step - loss: 0.1682 - accurac
y: 0.9417 - val_loss: 1.6412 - val_accuracy: 0.6845
Epoch 19/30
1563/1563 [==============================] - 216s 138ms/step - loss: 0.1514 - accurac
y: 0.9484 - val_loss: 1.5153 - val_accuracy: 0.6943
Epoch 20/30
1563/1563 [==============================] - 217s 139ms/step - loss: 0.1366 - accurac
y: 0.9541 - val_loss: 1.6062 - val_accuracy: 0.6847
Epoch 21/30
1563/1563 [==============================] - 216s 138ms/step - loss: 0.1378 - accurac
y: 0.9528 - val_loss: 1.6728 - val_accuracy: 0.6881
```

The speed of the train was very slow and after a while we encountered overfitting.



```
313/313 [==============================] - 9s 28ms/step - loss: 1.0853 - accuracy: 0.7016
```

Therefore, increasing the number of convolutional layers in models does not always create a better model, like our model, which has less accuracy in predicting test data. But it is more likely that in heavier models with more features, it will definitely be more useful to have more hidden layers.

Now we try the first model with batch normalization and dropout method to see the result.

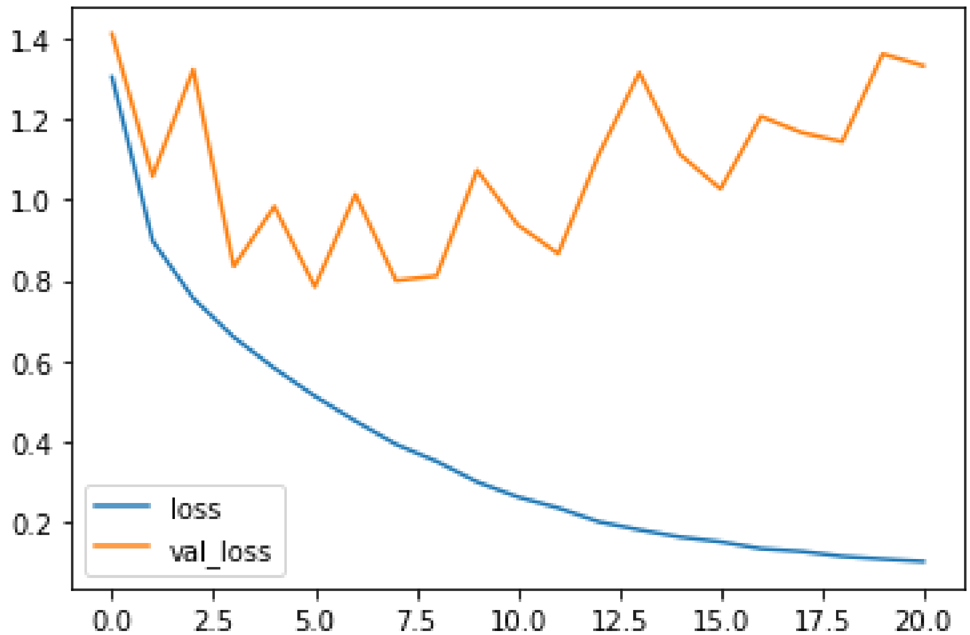
According to the extracted results, we understand that our model performs better training and performs better in predicting test data. And it even better presents the problem of other models, most of which was in distinguishing between dogs and cats

In [116]:

```python
input = Input(shape=(32, 32, 3))

#block1
x = Conv2D(filters=32, kernel_size=(5, 5), strides=1, padding='same')(input)
x = BatchNormalization()(x)
x = ReLU()(x)
x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

#block2
x = Conv2D(filters=64, kernel_size=(3, 3), strides=1, padding='same')(x)
x = BatchNormalization()(x)
x = ReLU()(x)
x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

#block3
x = Conv2D(filters=128, kernel_size=(3, 3), strides=1, padding='same')(x)
x = BatchNormalization()(x)
x = ReLU()(x)
x = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')(x)

#flatten
x = Flatten()(x)

# fully connected
x = Dense(units=128)(x)
x = ReLU()(x)
x = Dense(units=10)(x)
output = Activation(activation='softmax')(x)

cnn_model3 = Model(input, output)
cnn_model3.summary()
```

```
Epoch 1/30
1563/1563 [==============================] - 138s 88ms/step - loss: 1.3039 - accuracy: 0.5349 - val_loss: 1.4107 - val_accuracy: 0.5331
Epoch 2/30
1563/1563 [==============================] - 144s 92ms/step - loss: 0.8979 - accuracy: 0.6853 - val_loss: 1.0579 - val_accuracy: 0.6297
Epoch 3/30
1563/1563 [==============================] - 143s 92ms/step - loss: 0.7555 - accuracy: 0.7371 - val_loss: 1.3228 - val_accuracy: 0.5983
Epoch 4/30
1563/1563 [==============================] - 142s 91ms/step - loss: 0.6588 - accuracy: 0.7695 - val_loss: 0.8341 - val_accuracy: 0.7119
Epoch 5/30
```

```
1563/1563 [==============================] - 150s 96ms/step - loss: 0.5817 - accuracy
: 0.7968 - val_loss: 0.9824 - val_accuracy: 0.6633
Epoch 6/30
1563/1563 [==============================] - 144s 92ms/step - loss: 0.5123 - accuracy
: 0.8221 - val_loss: 0.7841 - val_accuracy: 0.7370
Epoch 7/30
1563/1563 [==============================] - 141s 90ms/step - loss: 0.4509 - accuracy
: 0.8414 - val_loss: 1.0123 - val_accuracy: 0.6766
Epoch 8/30
1563/1563 [==============================] - 143s 91ms/step - loss: 0.3931 - accuracy
: 0.8631 - val_loss: 0.7992 - val_accuracy: 0.7473
Epoch 9/30
1563/1563 [==============================] - 140s 90ms/step - loss: 0.3509 - accuracy
: 0.8758 - val_loss: 0.8105 - val_accuracy: 0.7499
Epoch 10/30
1563/1563 [==============================] - 137s 88ms/step - loss: 0.2998 - accuracy
: 0.8941 - val_loss: 1.0729 - val_accuracy: 0.6974
Epoch 11/30
1563/1563 [==============================] - 137s 88ms/step - loss: 0.2626 - accuracy
: 0.9080 - val_loss: 0.9369 - val_accuracy: 0.7372
Epoch 12/30
1563/1563 [==============================] - 136s 87ms/step - loss: 0.2352 - accuracy
: 0.9166 - val_loss: 0.8660 - val_accuracy: 0.7545
Epoch 13/30
1563/1563 [==============================] - 137s 87ms/step - loss: 0.2008 - accuracy
: 0.9297 - val_loss: 1.1113 - val_accuracy: 0.7263
Epoch 14/30
1563/1563 [==============================] - 139s 89ms/step - loss: 0.1808 - accuracy
: 0.9357 - val_loss: 1.3149 - val_accuracy: 0.6830
Epoch 15/30
1563/1563 [==============================] - 137s 88ms/step - loss: 0.1630 - accuracy
: 0.9418 - val_loss: 1.1120 - val_accuracy: 0.7264
Epoch 16/30
1563/1563 [==============================] - 138s 88ms/step - loss: 0.1512 - accuracy
: 0.9453 - val_loss: 1.0260 - val_accuracy: 0.7687
Epoch 17/30
1563/1563 [==============================] - 137s 88ms/step - loss: 0.1342 - accuracy
: 0.9521 - val_loss: 1.2058 - val_accuracy: 0.7479
Epoch 18/30
1563/1563 [==============================] - 137s 88ms/step - loss: 0.1272 - accuracy
: 0.9544 - val_loss: 1.1660 - val_accuracy: 0.7539
Epoch 19/30
1563/1563 [==============================] - 137s 88ms/step - loss: 0.1151 - accuracy
: 0.9580 - val_loss: 1.1434 - val_accuracy: 0.7497
Epoch 20/30
1563/1563 [==============================] - 137s 88ms/step - loss: 0.1083 - accuracy
: 0.9617 - val_loss: 1.3608 - val_accuracy: 0.7291
Epoch 21/30
1563/1563 [==============================] - 138s 88ms/step - loss: 0.1024 - accuracy
: 0.9638 - val_loss: 1.3316 - val_accuracy: 0.7507
```




```
313/313 [==============================] - 7s 21ms/step - loss: 1.0260 - accuracy: 0.
7687
```

Finally, we use the confusing matrix and classification report to see the details of the models more precisely. The comparison of three graphs of all three models is as follows.

Model 3:

In [42]:
```python
print(classification_report(test_y, predictions_sparse))
```

```
              precision    recall  f1-score   support

           0       0.83      0.72      0.77      1000
           1       0.83      0.85      0.84      1000
           2       0.61      0.62      0.62      1000
           3       0.53      0.57      0.55      1000
           4       0.72      0.63      0.67      1000
           5       0.56      0.72      0.63      1000
           6       0.82      0.75      0.78      1000
           7       0.79      0.75      0.77      1000
           8       0.83      0.84      0.84      1000
           9       0.81      0.78      0.80      1000

    accuracy                           0.72     10000
   macro avg       0.73      0.72      0.73     10000
weighted avg       0.73      0.72      0.73     10000
```

In [43]:
```python
confusion_matrix(test_y, predictions_sparse)
```

Out[43]:
```
array([[720,  17,  70,  22,  14,  14,  13,  12,  71,  47],
       [ 10, 855,  12,  10,   2,   7,   7,   1,  28,  68],
       [ 42,   7, 625,  77,  77,  78,  43,  32,  12,   7],
       [ 11,  14,  51, 569,  40, 215,  39,  36,  13,  12],
       [ 14,   1,  80,  93, 631,  78,  32,  58,   7,   6],
       [  6,   5,  43, 144,  27, 719,  16,  32,   2,   6],
       [  6,   7,  51,  91,  28,  50, 750,   6,   7,   4],
       [ 11,   3,  42,  35,  52,  91,   3, 751,   2,  10],
       [ 32,  31,  28,  16,   5,  15,   3,   7, 842,  21],
       [ 19,  94,  18,  24,   4,  19,   5,  11,  25, 781]], dtype=int64)
```

# Model 2:

```
In [70]: print(classification_report(test_y, predictions_sparse2))
```

```
              precision    recall  f1-score   support

           0       0.76      0.74      0.75      1000
           1       0.84      0.81      0.83      1000
           2       0.53      0.67      0.59      1000
           3       0.50      0.50      0.50      1000
           4       0.70      0.61      0.65      1000
           5       0.62      0.59      0.60      1000
           6       0.71      0.78      0.74      1000
           7       0.79      0.74      0.76      1000
           8       0.85      0.80      0.83      1000
           9       0.79      0.77      0.78      1000

    accuracy                           0.70     10000
   macro avg       0.71      0.70      0.70     10000
weighted avg       0.71      0.70      0.70     10000
```

```
In [71]: confusion_matrix(test_y, predictions_sparse2)
```

```
Out[71]: array([[743,   9,  96,  17,  14,   6,  14,   8,  49,  44],
       [ 11, 813,  15,  11,   2,   6,  10,   5,  34,  93],
       [ 47,   7, 671,  66,  66,  37,  77,  20,   4,   5],
       [ 22,  14, 114, 498,  39, 175,  87,  34,   8,   9],
       [ 19,   6, 133,  69, 607,  47,  60,  49,   5,   5],
       [  7,   7,  83, 166,  39, 593,  41,  53,   4,   7],
       [  5,   8,  58,  75,  27,  37, 779,   2,   6,   3],
       [ 16,   2,  52,  55,  62,  52,   8, 736,   3,  14],
       [ 77,  24,  25,  19,   8,   4,   9,   7, 804,  23],
       [ 32,  79,  30,  25,   5,   4,   8,  17,  28, 772]], dtype=int64)
```

# Model 3:

```
In [123]: print(classification_report(test_y, predictions_sparse3))
```

```
              precision    recall  f1-score   support

           0       0.75      0.82      0.79      1000
           1       0.90      0.85      0.87      1000
           2       0.78      0.59      0.67      1000
           3       0.63      0.55      0.59      1000
           4       0.66      0.83      0.74      1000
           5       0.67      0.71      0.69      1000
           6       0.78      0.87      0.82      1000
           7       0.83      0.77      0.80      1000
           8       0.87      0.87      0.87      1000
           9       0.85      0.83      0.84      1000

    accuracy                           0.77     10000
   macro avg       0.77      0.77      0.77     10000
weighted avg       0.77      0.77      0.77     10000
```

```
In [124]: confusion_matrix(test_y, predictions_sparse3)
```

```
Out[124]: array([[822,  14,  45,  11,  15,   6,  10,  11,  46,  20],
       [ 24, 849,   2,   8,   4,   2,   7,   1,  20,  83],
       [ 59,   4, 590,  53, 113,  57,  75,  38,  10,   1],
       [ 23,   3,  26, 548,  87, 173,  76,  36,  20,   8],
       [ 22,   3,  27,  34, 829,  24,  33,  21,   5,   2],
       [ 18,   4,  25, 117,  54, 709,  24,  39,   7,   3],
       [  3,   1,  18,  29,  45,  18, 873,   4,   6,   3],
       [ 16,   2,  15,  40,  96,  46,   8, 771,   2,   4],
       [ 61,  11,   8,  10,   4,   8,   5,   4, 870,  19],
       [ 45,  54,   3,  17,   8,   9,  12,   9,  17, 826]], dtype=int64)
```