Name: Yuhan Shao
Student Number: 1002281327
UTORid: shaoyuha

Assignment 5

1.
(a) Is it **NOT** a good idea to train a neural network to classify an input image into either one of the employees and an "unknown" category. Since by assumption, we only have a few, around 5 images, per employee in the training part, there is too few training data for each class, i.e.: too few features, therefore, this may lead to the test part not accurate and give the wrong classification.

(b) Explain the benefit of using TF-IDF over a present/absent representation:
Since TF-IDF is re-weighting the entries such that words that appears in many images (documents) are down weighted, when a word is presenting/absenting, then the entries are down/up weighted, i.e.: the TF-IDF algorithm is used to weigh a keyword in any content and assign the importance to that keyword based on the number of times it appears in the document. It is easier to extract the most descriptive features in the document and easily compute the similarity. In conclusion, the higher the TF-IDF score (weight), the rarer the word and vice versa.

(c) (d) (f) (g) [in the comment part] (h)

```
from keras import backend as K

K.set_image_data_format('channels_first')

from keras.models import model_from_json

from facenet import load_dataset, load_facenet, img_to_encoding

import cv2 as cv

import numpy as np

import os

from sklearn.cluster import KMeans



def get_embedding_path():
    """
    q1(c)
    """
    model = load_facenet()

    files = os.listdir("saved_faces")

    embedding, paths = [], []

    for path in files:

        image = cv.imread("saved_faces/" + path)

        encode = img_to_encoding(image, model)

        embedding.append(encode)

        paths.append(path)
    return embedding, paths

def kmean_cluster():
    """
    q1(d)
    """
    embedding, paths = get_embedding_path()
    # store the first element of the embedding list
    cluster = []
    for v in embedding:
        cluster.append(v[0])
    kmeans = KMeans(n_clusters=6, random_state=0).fit(cluster)
    kmean_label = kmeans.labels_
    kmean_center = kmeans.cluster_centers_

    return kmean_label, kmean_center, paths


def inverted_index():
    """
    q1(f)
    """
    kmean_label, kmean_center, paths = kmean_cluster()
    inverted_idx = {}
    for i in range(len(paths)):
        index = kmean_label[i]
        if index in inverted_index:
            inverted_idx[index].append(paths[i])
        else:
            inverted_idx[index] = [paths[i]]

    return inverted_idx   # {label(0~5): [list of path]}


# q1(g):
# For each image in input_faces, you want to find it’s matching images from
# saved_faces. Describe a method to do this:
```

```python
# 1. get the list of embeddings and paths of the input images
# 2. for each input image:
#   -> get its embedding vector
#   -> Compute the similarity by normalized dot product between its embedding
#      vector and the position of the center of each cluster, say: sim(v, c)
#   -> Let the maximum sim(v, c) as the score of current image file
# 3. In order to get rid of some mismatch, given a threshold to filter the
#    the possible mismatching
#   -> For input image with similarity <= 0.8, identify it as the 'None' type
#   -> For input image with similarity > 0.8:
#      use the inverted_index() as helper function to identify which class
#      it belongs to.
# 4. Print out the classification in order

def loading_input():
    """
    q1(h) helper function, return the (embedding, paths) of the input file
    """
    model = load_facenet()
    input_img = os.listdir("input_faces")
    embedding, paths = [], []
    for path in input_img:
        image = cv.imread("input_faces/" + path)
        image = cv.resize(image, (96, 96))
        encode = img_to_encoding(image, model)
        embedding.append(encode)
        paths.append(path)

    return embedding, paths


def matching_algo():
    """
    q1(h)
    """
    dataset_inverted_index = inverted_index()
    _, kmean_center, _ = kmean_cluster()
```

```
input_embedding, input_paths = loading_input()

# calculate the max sim(t, v) of each input image file
image_sim = {}
for i in range(len(input_paths)):
    vector = input_embedding[i]
    possible_sim = []
    for j in range(len(kmean_center)):
        center = kmean_center[j]
        sim = np.divide(np.dot(vector, center), (np.linalg.norm(vector) * np.linalg.norm(center)))
        possible_sim.append([sim, j])
    image_sim[input_paths[i]] = max(possible_sim)  # (max_sim, center)

# threshold to filter the mismatching
threshold = 0.8
for name in image_sim:
    if image_sim[name][0] <= threshold:
        image_sim[name][1] = 'None'
    else:
        center_index = image_sim[name][1]
        target = dataset_inverted_index[center_index]
        image_sim[name][0] = target

# print out the final classification
for name in image_sim:
    print("Input image: " + name)
    print(" belongs to class: ")
    print(image_sim[name][1])
    print("==============================")

# the output
output = {}
for name in image_sim:
    if image_sim[name][1] != 'None':
        category = dataset_inverted_index[image_sim[name][1]]
        output[name] = category
    else:
            output[name] = []
    print(output)



if __name__ == "__main__":
    matching_algo()
```

(g) **The output of the part (g):**

```
Input image: face_image_107.jpg
 belongs to class:
3
==============================
Input image: face_image_109.jpg
 belongs to class:
4
==============================
Input image: face_image_116.jpg
 belongs to class:
1
==============================
Input image: face_image_119.jpg
 belongs to class:
5
==============================
Input image: face_image_126.jpg
 belongs to class:
4
==============================
Input image: face_image_600.jpg
 belongs to class:
None
==============================
Input image: face_image_97.jpg
 belongs to class:
5
==============================
{'face_image_107.jpg': ['face_image_112.jpg', 'face_image_115.jpg', 'face_image_117.jpg', 'face_image_130.jpg', 'face_image_133.jpg',
'face_image_138.jpg', 'face_image_143.jpg', 'face_image_144.jpg', 'face_image_145.jpg', 'face_image_149.jpg', 'face_image_15.jpg',
'face_image_154.jpg', 'face_image_157.jpg', 'face_image_158.jpg', 'face_image_160.jpg', 'face_image_175.jpg', 'face_image_187.jpg',
'face_image_188.jpg', 'face_image_19.jpg', 'face_image_190.jpg', 'face_image_30.jpg', 'face_image_31.jpg', 'face_image_32.jpg',
'face_image_38.jpg', 'face_image_4.jpg', 'face_image_47.jpg', 'face_image_50.jpg', 'face_image_58.jpg', 'face_image_6.jpg',
'face_image_69.jpg', 'face_image_7.jpg', 'face_image_70.jpg', 'face_image_81.jpg', 'face_image_87.jpg', 'face_image_9.jpg',
'face_image_91.jpg', 'face_image_95.jpg', 'face_image_96.jpg'], 'face_image_109.jpg': ['face_image_104.jpg', 'face_image_109.jpg',
'face_image_114.jpg', 'face_image_120.jpg', 'face_image_123.jpg', 'face_image_124.jpg', 'face_image_125.jpg', 'face_image_131.jpg',
'face_image_139.jpg', 'face_image_140.jpg', 'face_image_146.jpg', 'face_image_150.jpg', 'face_image_152.jpg', 'face_image_164.jpg',
'face_image_166.jpg', 'face_image_168.jpg', 'face_image_169.jpg', 'face_image_171.jpg', 'face_image_174.jpg', 'face_image_176.jpg',
'face_image_179.jpg', 'face_image_182.jpg', 'face_image_189.jpg', 'face_image_193.jpg', 'face_image_21.jpg', 'face_image_22.jpg',
'face_image_24.jpg', 'face_image_25.jpg', 'face_image_29.jpg', 'face_image_3.jpg', 'face_image_35.jpg', 'face_image_41.jpg',
'face_image_51.jpg', 'face_image_71.jpg', 'face_image_78.jpg'], 'face_image_116.jpg': ['face_image_1.jpg', 'face_image_11.jpg',
'face_image_116.jpg', 'face_image_14.jpg', 'face_image_148.jpg', 'face_image_16.jpg', 'face_image_162.jpg', 'face_image_172.jpg',
'face_image_177.jpg', 'face_image_184.jpg', 'face_image_196.jpg', 'face_image_2.jpg', 'face_image_23.jpg', 'face_image_36.jpg',
'face_image_46.jpg', 'face_image_48.jpg', 'face_image_5.jpg', 'face_image_55.jpg', 'face_image_56.jpg', 'face_image_61.jpg',
'face_image_66.jpg', 'face_image_79.jpg', 'face_image_88.jpg'], 'face_image_119.jpg': ['face_image_10.jpg', 'face_image_102.jpg',
'face_image_103.jpg', 'face_image_107.jpg', 'face_image_111.jpg', 'face_image_113.jpg', 'face_image_12.jpg', 'face_image_13.jpg',
'face_image_132.jpg', 'face_image_136.jpg', 'face_image_142.jpg', 'face_image_170.jpg', 'face_image_178.jpg', 'face_image_183.jpg',
'face_image_191.jpg', 'face_image_26.jpg', 'face_image_39.jpg', 'face_image_42.jpg', 'face_image_43.jpg', 'face_image_53.jpg',
'face_image_59.jpg', 'face_image_63.jpg', 'face_image_64.jpg', 'face_image_65.jpg', 'face_image_68.jpg', 'face_image_72.jpg',
'face_image_73.jpg', 'face_image_75.jpg', 'face_image_85.jpg', 'face_image_94.jpg', 'face_image_97.jpg'], 'face_image_126.jpg':
['face_image_104.jpg', 'face_image_109.jpg', 'face_image_114.jpg', 'face_image_120.jpg', 'face_image_123.jpg', 'face_image_124.jpg',
'face_image_125.jpg', 'face_image_131.jpg', 'face_image_139.jpg', 'face_image_140.jpg', 'face_image_146.jpg', 'face_image_150.jpg',
'face_image_152.jpg', 'face_image_164.jpg', 'face_image_166.jpg', 'face_image_168.jpg', 'face_image_169.jpg', 'face_image_171.jpg',
'face_image_174.jpg', 'face_image_176.jpg', 'face_image_179.jpg', 'face_image_182.jpg', 'face_image_189.jpg', 'face_image_193.jpg',
'face_image_21.jpg', 'face_image_22.jpg', 'face_image_24.jpg', 'face_image_25.jpg', 'face_image_29.jpg', 'face_image_3.jpg',
'face_image_35.jpg', 'face_image_41.jpg', 'face_image_51.jpg', 'face_image_71.jpg', 'face_image_78.jpg'], 'face_image_600.jpg': [],
'face_image_97.jpg': ['face_image_10.jpg', 'face_image_102.jpg', 'face_image_103.jpg', 'face_image_107.jpg', 'face_image_111.jpg',
'face_image_113.jpg', 'face_image_12.jpg', 'face_image_13.jpg', 'face_image_132.jpg', 'face_image_136.jpg', 'face_image_142.jpg',
'face_image_170.jpg', 'face_image_178.jpg', 'face_image_183.jpg', 'face_image_191.jpg', 'face_image_26.jpg', 'face_image_39.jpg',
'face_image_42.jpg', 'face_image_43.jpg', 'face_image_53.jpg', 'face_image_59.jpg', 'face_image_63.jpg', 'face_image_64.jpg',
'face_image_65.jpg', 'face_image_68.jpg', 'face_image_72.jpg', 'face_image_73.jpg', 'face_image_75.jpg', 'face_image_85.jpg',
'face_image_94.jpg', 'face_image_97.jpg']}
```

(i)A more general retrieval system: (based on embedding)
→ First, suppose there are **n employees**, considered as the dataset. Similarly, use the 'inverted index' method as in the part (f), for each of the n employees.
→ Then, there is an input image with the target employee and other people who is possibly an employee or the other people who is not a 'class' that is included in the 'inverted index' dictionary.
→ Use the 'Bounding Box' method, sliding through the image, find all the faces.
→ Loop through all the faces, **for each face, calculate the similarity (same as the part(h)) of current face and the n employees in the 'inverted index' dictionary**. **(*)**

- ➔ Pick the bounding box with the highest similarity/score.
- ➔ Given **a threshold**: if the similarity of this bounding box is lower than threshold, give 'None', i.e.: this face is not a recorded employee. Else, this is the employee we wanted.

■ **Challenges:**

(1) If the number of employees who need to be recorded in too large and the number of faces detected by the bounding box too large, then, there may be the running time issue in (*)
(2) Also, the choice of the threshold may affect the final output.

2.
(a)

■ Describe the vanishing gradient problem:

When training artificial neural networks with gradient-based learning methods and backpropagation, each of the neural network's weights receives an update proportional to the partial derivative of the error function with respect to the current weight in each iteration of training. **Problem comes that** in some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value. In the worst case, this may completely stop the neural network from further training.

■ Some ways to mitigate the vanishing gradient problem:

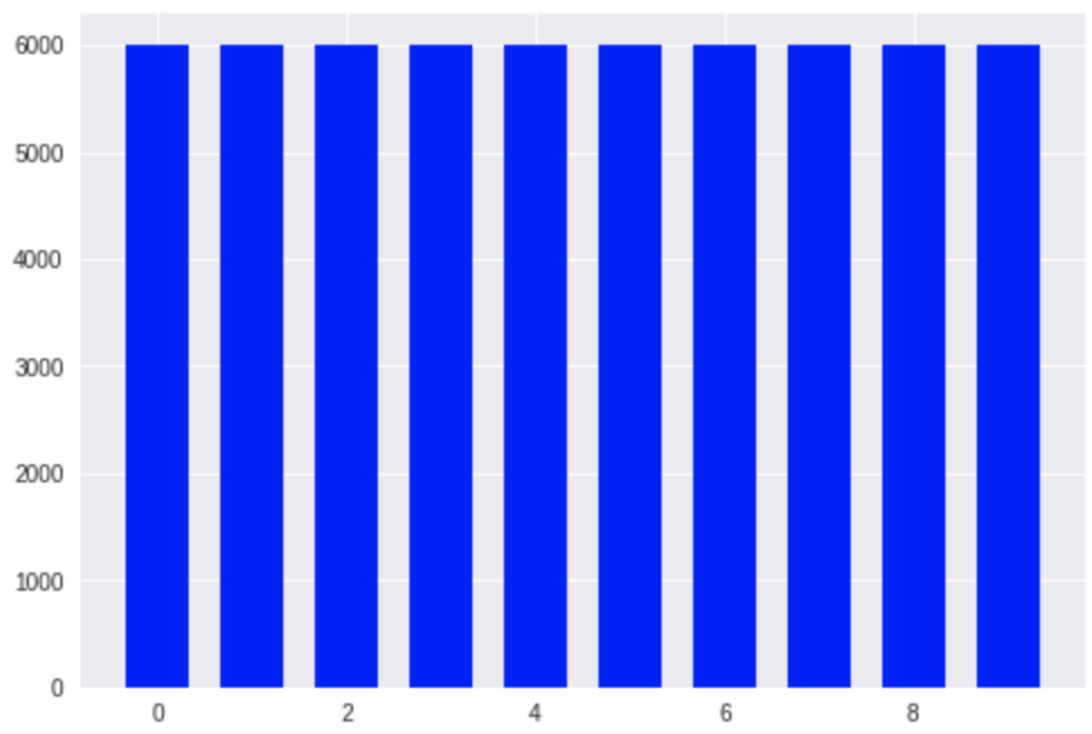
1. Multi-level hierarchy: pre-trained one level at a time through unsupervised learning.
2. Residual networks: splitting a deep network into three layer chunks and passing the input into each chunk straight through to the next chunk, along with the residual-output of the chunk minus the input to the chunk that is reintroduced.

(b) CNN's include max pooling layers since we want to get invariance to small shifts in position. We use pooling layer in order to reduce the number of parameters and amount of computations in the network, and hence to also control overfitting.

(c) (i)

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



# loading the data
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# store class name for future use
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# show a bar graph of number of instance in each class
class_dict = {}
for i in range(len(train_labels)):
    if train_labels[i] in class_dict:
        class_dict[train_labels[i]] += 1
    else:
        class_dict[train_labels[i]] = 1

# show a bar graph of the number of instance in each class
class_lst = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
num_instance = []
for i in range(len(class_lst)):
    num_instance.append(class_dict[class_lst[i]])

width = 1/1.5

plt.bar(class_lst, num_instance, width, color="blue")
```

■ Bar output:



(ii)

```
# split the dataset into training set (70) and validation set (30)
# 1. according to the training label, split the training data into the 10 classes
split_class = {}  # dict of {class_num: [list of img]}
for i in range(len(train_labels)):
    if train_labels[i] in split_class:
        split_class[train_labels[i]].append(train_images[i])
    else:
        split_class[train_labels[i]] = [train_images[i]]

# 2. for each class, first random the image data in the list, then split to 7:3
img_class = split_class.values()  # list of lists of data
training_set, validation_set = [], []
for item_class in img_class:
    random.shuffle(item_class)
    training_set.extend(item_class[:4200])
    validation_set.extend(item_class[4200:])
```

(d) Build up the model using ‘tensorflow’ and ‘keras’

```
# build up the model
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

- ■ What should the activation at the output be:
- The 'activation' of the first Dense layer: **tf.nn.relu:**
  Computes rectified linear: max(features, 0)
- The 'activation' of the first Dense layer: **tf.nn.softmax:**
  Computes softmax activations. Equivalent to:
  softmax = tf.exp(logits) / tf.reduce_sum(tf.exp(logits), axis)

(e) – The lose function is: 'sparse_categorical_crossentropy'

'crossentropy' is a loss function, used to measure the dissimilarity between the distribution of observed class labels and the predicted probabilities of class membership.
'categorical' refers to the possibility of having more than two classes (instead of binary, which refers to two classes).
'sparse' means that it does use all the possible classes but some of them.

(f) What is your choice of batch size? When Should you stop training?
- The batch size is 16.
- In order to prevent the overfitting, we need to stop training when the validation error is the minimum.

Learning Rate	Batch Size	Number of Epochs trained	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
None	16	15	0.1996	0.9247	0.3374	0.8872

```
# build up the model
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
history = model.fit(train_images, train_labels, epochs=15, batch_size=16, validation_split=0.3)
```

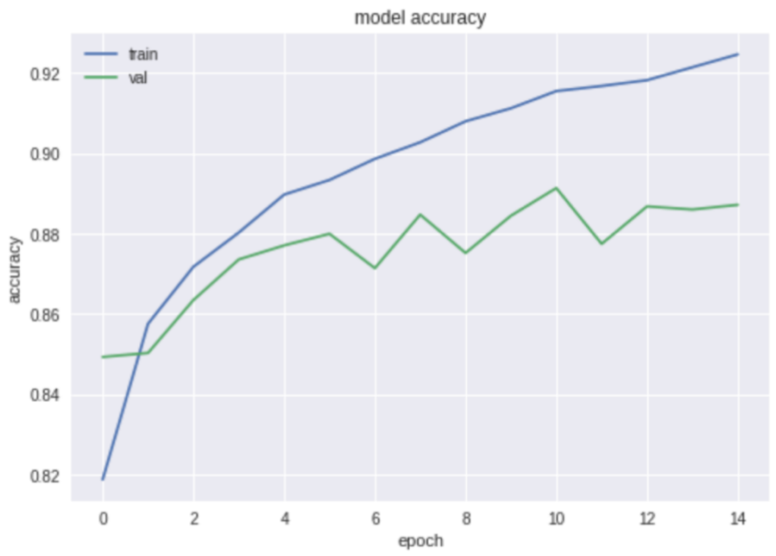
■ Output:

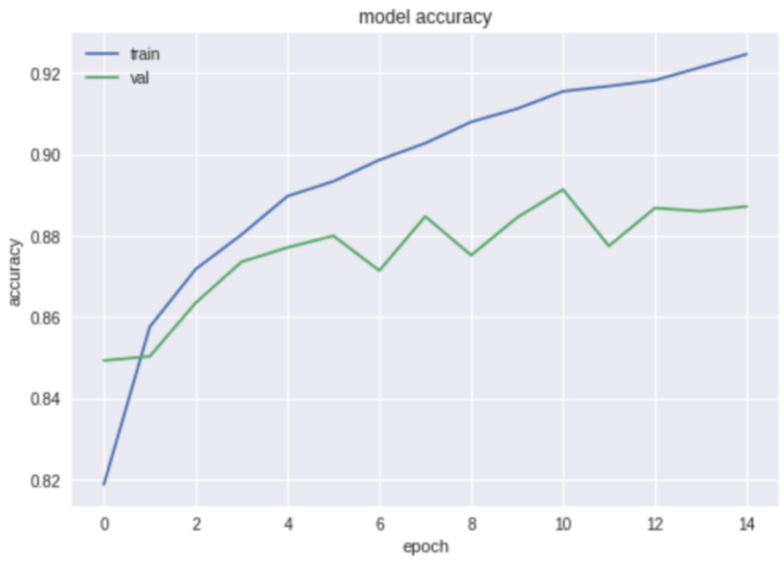
```
Train on 42000 samples, validate on 18000 samples
Epoch 1/15
42000/42000 [==============================] - 7s 172us/step - loss: 0.5113 - acc: 0.8188 - val_loss: 0.4182 - val_acc: 0.8493
Epoch 2/15
42000/42000 [==============================] - 7s 162us/step - loss: 0.3871 - acc: 0.8576 - val_loss: 0.4152 - val_acc: 0.8503
Epoch 3/15
42000/42000 [==============================] - 7s 167us/step - loss: 0.3460 - acc: 0.8718 - val_loss: 0.3822 - val_acc: 0.8635
Epoch 4/15
42000/42000 [==============================] - 7s 167us/step - loss: 0.3217 - acc: 0.8803 - val_loss: 0.3482 - val_acc: 0.8736
Epoch 5/15
42000/42000 [==============================] - 8s 189us/step - loss: 0.2982 - acc: 0.8898 - val_loss: 0.3453 - val_acc: 0.8771
Epoch 6/15
42000/42000 [==============================] - 7s 160us/step - loss: 0.2847 - acc: 0.8934 - val_loss: 0.3341 - val_acc: 0.8800
Epoch 7/15
42000/42000 [==============================] - 7s 164us/step - loss: 0.2700 - acc: 0.8987 - val_loss: 0.3550 - val_acc: 0.8714
Epoch 8/15
42000/42000 [==============================] - 7s 158us/step - loss: 0.2584 - acc: 0.9028 - val_loss: 0.3238 - val_acc: 0.8848
Epoch 9/15
42000/42000 [==============================] - 7s 161us/step - loss: 0.2461 - acc: 0.9080 - val_loss: 0.3706 - val_acc: 0.8752
Epoch 10/15
42000/42000 [==============================] - 7s 167us/step - loss: 0.2369 - acc: 0.9113 - val_loss: 0.3312 - val_acc: 0.8845
Epoch 11/15
42000/42000 [==============================] - 7s 170us/step - loss: 0.2277 - acc: 0.9155 - val_loss: 0.3214 - val_acc: 0.8914
Epoch 12/15
42000/42000 [==============================] - 7s 166us/step - loss: 0.2219 - acc: 0.9168 - val_loss: 0.3581 - val_acc: 0.8775
Epoch 13/15
42000/42000 [==============================] - 7s 164us/step - loss: 0.2159 - acc: 0.9183 - val_loss: 0.3326 - val_acc: 0.8868
Epoch 14/15
42000/42000 [==============================] - 7s 165us/step - loss: 0.2081 - acc: 0.9215 - val_loss: 0.3400 - val_acc: 0.8861
Epoch 15/15
42000/42000 [==============================] - 8s 181us/step - loss: 0.1996 - acc: 0.9247 - val_loss: 0.3374 - val_acc: 0.8872
```

■ Plotting:

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

■ Diagram





(g)

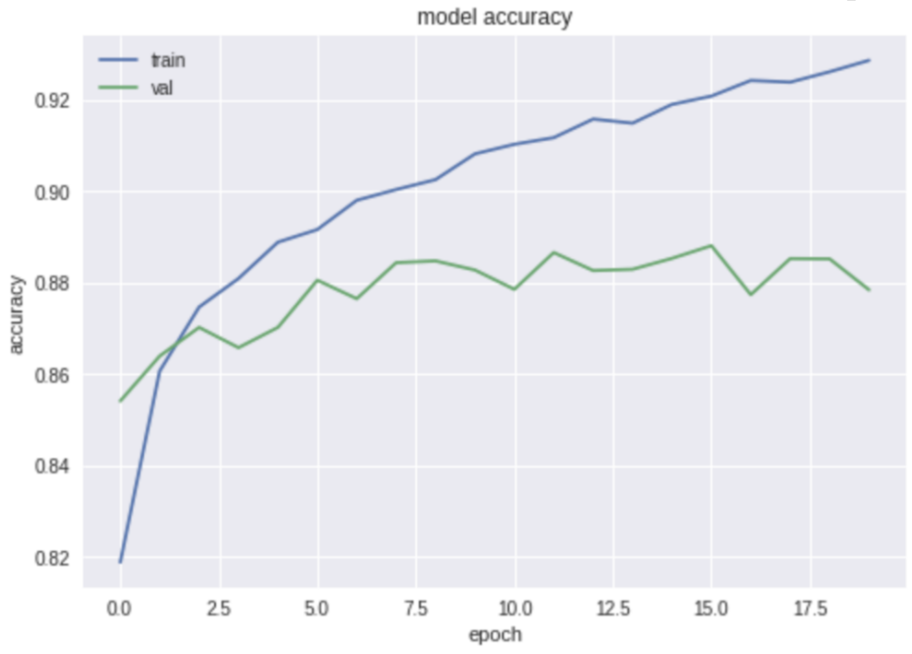
■ **Data plotting code:**

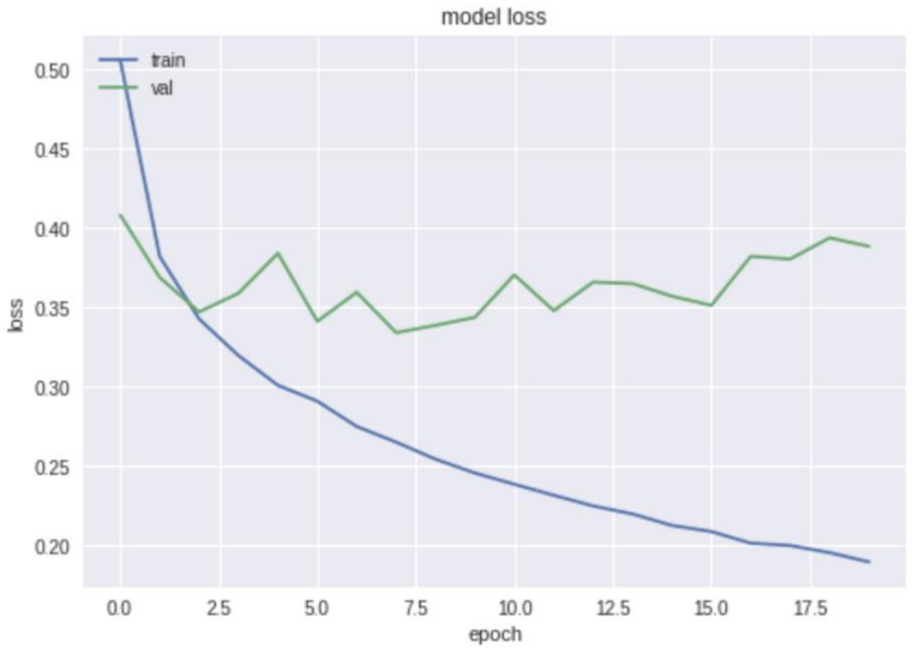
```
# changing batch size
n_batch = [8, 16, 32, 64]
for i in n_batch:
    model = keras.Sequential([
        keras.layers.Flatten(input_shape=(28, 28)),
        keras.layers.Dense(128, activation=tf.nn.relu),
        keras.layers.Dense(10, activation=tf.nn.softmax)
    ])
    model.compile(optimizer=tf.train.AdamOptimizer(),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    history = model.fit(train_images, train_labels, epochs=20, batch_size=i, validation_split=0.3)
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
    model.reset_states()

train_loss, train_acc = model.evaluate(train_images, train_labels)
test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)
```

■ **The output diagram of each batch size:**

```
Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 14s 331us/step - loss: 0.5056 - acc: 0.8188 - val_loss: 0.4081 - val_acc: 0.8541
Epoch 2/20
42000/42000 [==============================] - 13s 316us/step - loss: 0.3821 - acc: 0.8607 - val_loss: 0.3688 - val_acc: 0.8639
Epoch 3/20
42000/42000 [==============================] - 13s 313us/step - loss: 0.3426 - acc: 0.8747 - val_loss: 0.3471 - val_acc: 0.8702
Epoch 4/20
42000/42000 [==============================] - 13s 313us/step - loss: 0.3196 - acc: 0.8809 - val_loss: 0.3588 - val_acc: 0.8658
Epoch 5/20
42000/42000 [==============================] - 13s 317us/step - loss: 0.3008 - acc: 0.8889 - val_loss: 0.3841 - val_acc: 0.8702
Epoch 6/20
42000/42000 [==============================] - 14s 343us/step - loss: 0.2908 - acc: 0.8916 - val_loss: 0.3412 - val_acc: 0.8806
Epoch 7/20
42000/42000 [==============================] - 13s 313us/step - loss: 0.2749 - acc: 0.8980 - val_loss: 0.3594 - val_acc: 0.8765
Epoch 8/20
42000/42000 [==============================] - 13s 312us/step - loss: 0.2650 - acc: 0.9004 - val_loss: 0.3340 - val_acc: 0.8844
Epoch 9/20
42000/42000 [==============================] - 13s 306us/step - loss: 0.2543 - acc: 0.9026 - val_loss: 0.3386 - val_acc: 0.8848
Epoch 10/20
42000/42000 [==============================] - 13s 315us/step - loss: 0.2455 - acc: 0.9082 - val_loss: 0.3437 - val_acc: 0.8828
Epoch 11/20
42000/42000 [==============================] - 13s 315us/step - loss: 0.2384 - acc: 0.9103 - val_loss: 0.3705 - val_acc: 0.8786
Epoch 12/20
42000/42000 [==============================] - 13s 310us/step - loss: 0.2315 - acc: 0.9118 - val_loss: 0.3479 - val_acc: 0.8866
Epoch 13/20
42000/42000 [==============================] - 13s 307us/step - loss: 0.2248 - acc: 0.9158 - val_loss: 0.3659 - val_acc: 0.8827
Epoch 14/20
42000/42000 [==============================] - 13s 307us/step - loss: 0.2197 - acc: 0.9149 - val_loss: 0.3649 - val_acc: 0.8829
Epoch 15/20
42000/42000 [==============================] - 13s 314us/step - loss: 0.2125 - acc: 0.9190 - val_loss: 0.3570 - val_acc: 0.8853
Epoch 16/20
42000/42000 [==============================] - 13s 317us/step - loss: 0.2086 - acc: 0.9209 - val_loss: 0.3512 - val_acc: 0.8881
Epoch 17/20
```

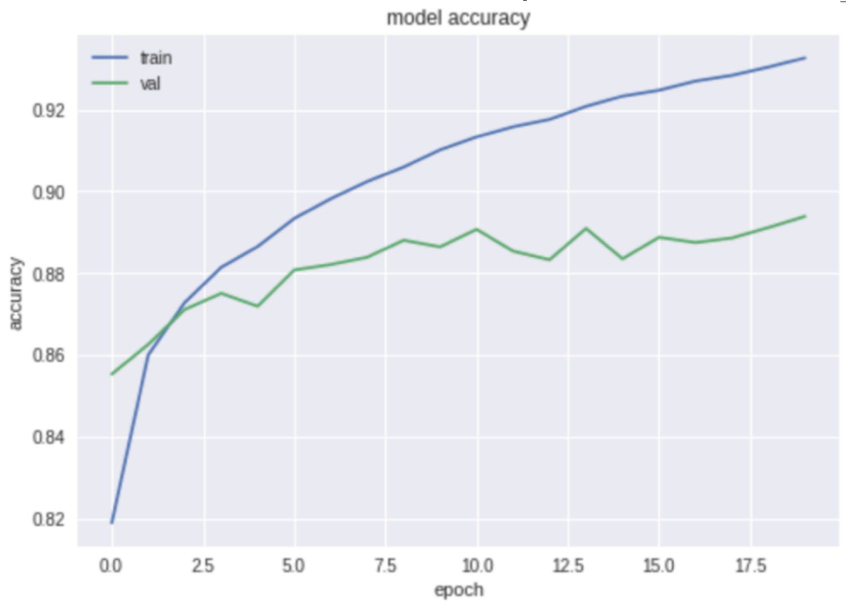
```
42000/42000 [==============================] - 13s 309us/step - loss: 0.2013 - acc: 0.9243 - val_loss: 0.3822 - val_acc: 0.8774
Epoch 18/20
42000/42000 [==============================] - 13s 313us/step - loss: 0.1998 - acc: 0.9239 - val_loss: 0.3804 - val_acc: 0.8853
Epoch 19/20
42000/42000 [==============================] - 13s 314us/step - loss: 0.1954 - acc: 0.9262 - val_loss: 0.3938 - val_acc: 0.8852
Epoch 20/20
42000/42000 [==============================] - 13s 310us/step - loss: 0.1895 - acc: 0.9287 - val_loss: 0.3884 - val_acc: 0.8784
```

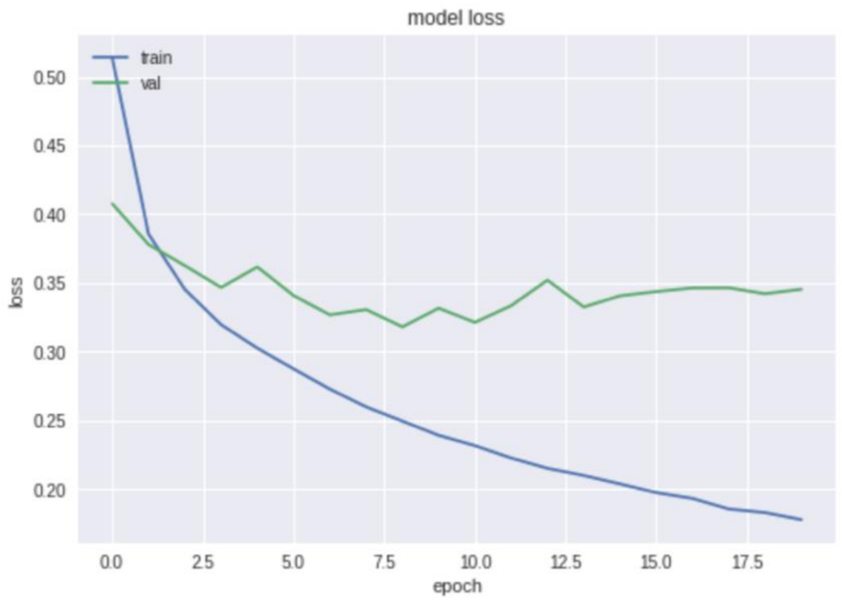




```
Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 9s 206us/step - loss: 0.5139 - acc: 0.8189 - val_loss: 0.4077 - val_acc: 0.8553
Epoch 2/20
42000/42000 [==============================] - 8s 192us/step - loss: 0.3857 - acc: 0.8600 - val_loss: 0.3780 - val_acc: 0.8626
Epoch 3/20
42000/42000 [==============================] - 7s 172us/step - loss: 0.3453 - acc: 0.8728 - val_loss: 0.3627 - val_acc: 0.8711
Epoch 4/20
42000/42000 [==============================] - 7s 171us/step - loss: 0.3197 - acc: 0.8814 - val_loss: 0.3467 - val_acc: 0.8751
Epoch 5/20
42000/42000 [==============================] - 7s 173us/step - loss: 0.3025 - acc: 0.8865 - val_loss: 0.3617 - val_acc: 0.8719
Epoch 6/20
42000/42000 [==============================] - 7s 176us/step - loss: 0.2874 - acc: 0.8934 - val_loss: 0.3408 - val_acc: 0.8808
Epoch 7/20
42000/42000 [==============================] - 7s 175us/step - loss: 0.2727 - acc: 0.8982 - val_loss: 0.3268 - val_acc: 0.8821
Epoch 8/20
42000/42000 [==============================] - 7s 175us/step - loss: 0.2598 - acc: 0.9024 - val_loss: 0.3306 - val_acc: 0.8839
```

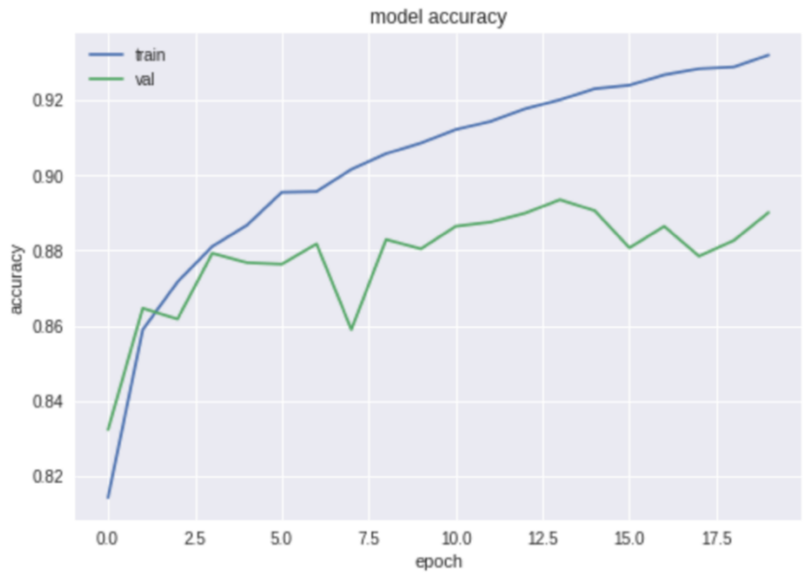
```
Epoch 9/20
42000/42000 [==============================] - 7s 174us/step - loss: 0.2494 - acc: 0.9060 - val_loss: 0.3180 - val_acc: 0.8881
Epoch 10/20
42000/42000 [==============================] - 7s 172us/step - loss: 0.2391 - acc: 0.9102 - val_loss: 0.3316 - val_acc: 0.8864
Epoch 11/20
42000/42000 [==============================] - 7s 172us/step - loss: 0.2316 - acc: 0.9134 - val_loss: 0.3213 - val_acc: 0.8907
Epoch 12/20
42000/42000 [==============================] - 7s 173us/step - loss: 0.2225 - acc: 0.9158 - val_loss: 0.3335 - val_acc: 0.8854
Epoch 13/20
42000/42000 [==============================] - 7s 174us/step - loss: 0.2150 - acc: 0.9176 - val_loss: 0.3520 - val_acc: 0.8833
Epoch 14/20
42000/42000 [==============================] - 7s 175us/step - loss: 0.2098 - acc: 0.9208 - val_loss: 0.3325 - val_acc: 0.8909
Epoch 15/20
42000/42000 [==============================] - 7s 175us/step - loss: 0.2037 - acc: 0.9233 - val_loss: 0.3406 - val_acc: 0.8835
Epoch 16/20
42000/42000 [==============================] - 8s 201us/step - loss: 0.1974 - acc: 0.9248 - val_loss: 0.3437 - val_acc: 0.8888
Epoch 17/20
42000/42000 [==============================] - 7s 172us/step - loss: 0.1931 - acc: 0.9270 - val_loss: 0.3464 - val_acc: 0.8875
Epoch 18/20
42000/42000 [==============================] - 7s 175us/step - loss: 0.1854 - acc: 0.9284 - val_loss: 0.3465 - val_acc: 0.8886
Epoch 19/20
42000/42000 [==============================] - 7s 175us/step - loss: 0.1828 - acc: 0.9305 - val_loss: 0.3421 - val_acc: 0.8912
Epoch 20/20
42000/42000 [==============================] - 7s 172us/step - loss: 0.1777 - acc: 0.9327 - val_loss: 0.3454 - val_acc: 0.8939
```

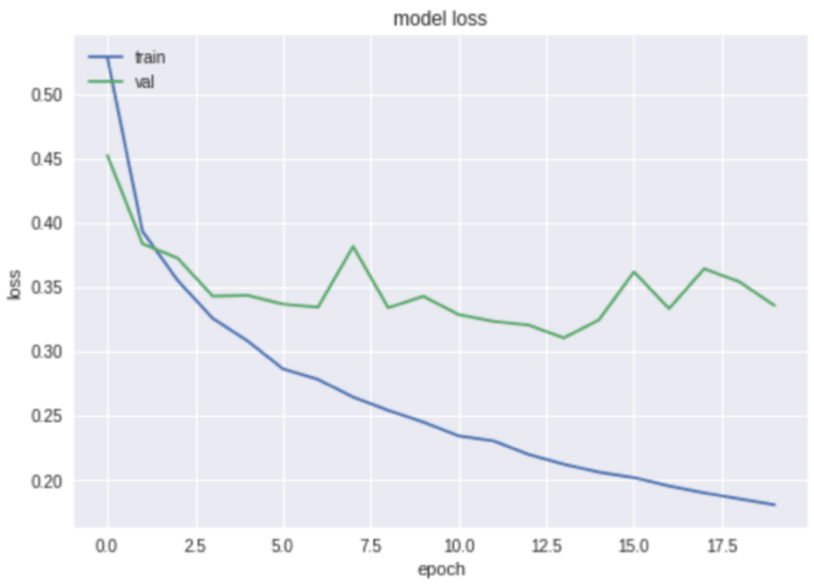




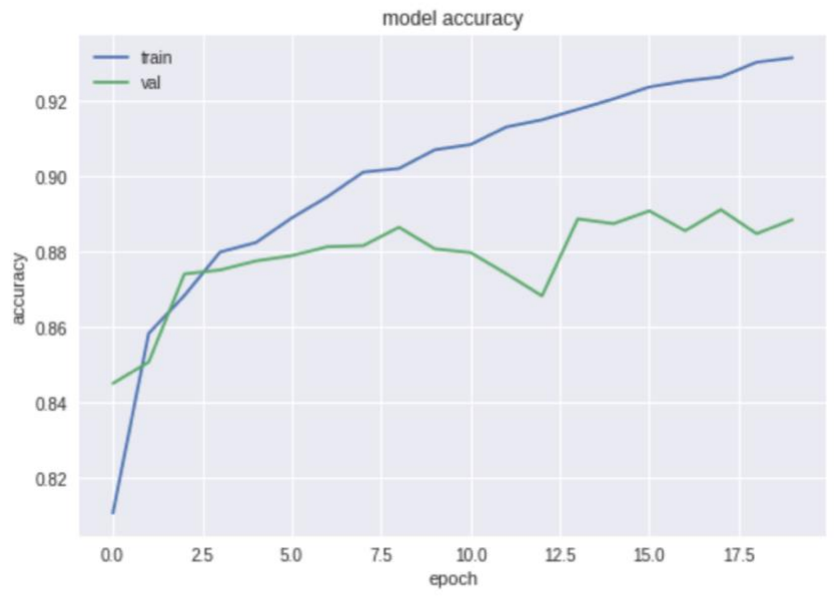
```
Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 5s 120us/step - loss: 0.5283 - acc: 0.8143 - val_loss: 0.4522 - val_acc: 0.8323
Epoch 2/20
42000/42000 [==============================] - 4s 107us/step - loss: 0.3933 - acc: 0.8590 - val_loss: 0.3836 - val_acc: 0.8647
```

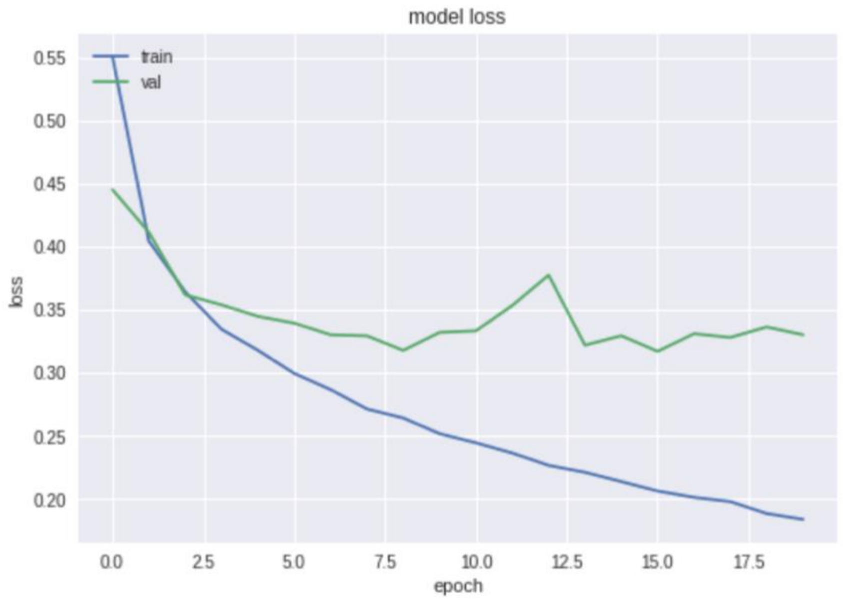
```
Epoch 3/20
42000/42000 [==============================] - 5s 107us/step - loss: 0.3551 - acc: 0.8717 - val_loss: 0.3724 - val_acc: 0.8618
Epoch 4/20
42000/42000 [==============================] - 4s 105us/step - loss: 0.3256 - acc: 0.8811 - val_loss: 0.3428 - val_acc: 0.8793
Epoch 5/20
42000/42000 [==============================] - 5s 108us/step - loss: 0.3079 - acc: 0.8868 - val_loss: 0.3434 - val_acc: 0.8768
Epoch 6/20
42000/42000 [==============================] - 5s 108us/step - loss: 0.2864 - acc: 0.8955 - val_loss: 0.3367 - val_acc: 0.8764
Epoch 7/20
42000/42000 [==============================] - 4s 107us/step - loss: 0.2782 - acc: 0.8957 - val_loss: 0.3342 - val_acc: 0.8818
Epoch 8/20
42000/42000 [==============================] - 5s 108us/step - loss: 0.2645 - acc: 0.9016 - val_loss: 0.3815 - val_acc: 0.8590
Epoch 9/20
42000/42000 [==============================] - 5s 108us/step - loss: 0.2542 - acc: 0.9058 - val_loss: 0.3338 - val_acc: 0.8829
Epoch 10/20
42000/42000 [==============================] - 4s 106us/step - loss: 0.2450 - acc: 0.9086 - val_loss: 0.3428 - val_acc: 0.8804
Epoch 11/20
42000/42000 [==============================] - 4s 105us/step - loss: 0.2344 - acc: 0.9121 - val_loss: 0.3286 - val_acc: 0.8864
Epoch 12/20
42000/42000 [==============================] - 4s 106us/step - loss: 0.2304 - acc: 0.9143 - val_loss: 0.3233 - val_acc: 0.8876
Epoch 13/20
42000/42000 [==============================] - 5s 108us/step - loss: 0.2198 - acc: 0.9177 - val_loss: 0.3204 - val_acc: 0.8899
Epoch 14/20
42000/42000 [==============================] - 5s 109us/step - loss: 0.2123 - acc: 0.9201 - val_loss: 0.3104 - val_acc: 0.8935
Epoch 15/20
42000/42000 [==============================] - 5s 108us/step - loss: 0.2061 - acc: 0.9230 - val_loss: 0.3244 - val_acc: 0.8906
Epoch 16/20
42000/42000 [==============================] - 5s 109us/step - loss: 0.2019 - acc: 0.9240 - val_loss: 0.3617 - val_acc: 0.8807
Epoch 17/20
42000/42000 [==============================] - 4s 107us/step - loss: 0.1953 - acc: 0.9267 - val_loss: 0.3332 - val_acc: 0.8864
Epoch 18/20
42000/42000 [==============================] - 4s 106us/step - loss: 0.1901 - acc: 0.9283 - val_loss: 0.3641 - val_acc: 0.8785
Epoch 19/20
42000/42000 [==============================] - 5s 108us/step - loss: 0.1854 - acc: 0.9288 - val_loss: 0.3540 - val_acc: 0.8827
Epoch 20/20
42000/42000 [==============================] - 5s 108us/step - loss: 0.1808 - acc: 0.9320 - val_loss: 0.3357 - val_acc: 0.8901
```





```
Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 3s 81us/step - loss: 0.5504 - acc: 0.8107 - val_loss: 0.4451 - val_acc: 0.8452
Epoch 2/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.4043 - acc: 0.8584 - val_loss: 0.4113 - val_acc: 0.8508
Epoch 3/20
42000/42000 [==============================] - 3s 66us/step - loss: 0.3641 - acc: 0.8685 - val_loss: 0.3617 - val_acc: 0.8742
Epoch 4/20
42000/42000 [==============================] - 3s 66us/step - loss: 0.3344 - acc: 0.8800 - val_loss: 0.3538 - val_acc: 0.8753
Epoch 5/20
42000/42000 [==============================] - 3s 65us/step - loss: 0.3178 - acc: 0.8825 - val_loss: 0.3447 - val_acc: 0.8777
Epoch 6/20
42000/42000 [==============================] - 3s 66us/step - loss: 0.2995 - acc: 0.8891 - val_loss: 0.3393 - val_acc: 0.8791
Epoch 7/20
42000/42000 [==============================] - 3s 65us/step - loss: 0.2867 - acc: 0.8947 - val_loss: 0.3301 - val_acc: 0.8814
Epoch 8/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.2713 - acc: 0.9012 - val_loss: 0.3292 - val_acc: 0.8817
Epoch 9/20
42000/42000 [==============================] - 3s 65us/step - loss: 0.2641 - acc: 0.9022 - val_loss: 0.3177 - val_acc: 0.8866
Epoch 10/20
42000/42000 [==============================] - 3s 66us/step - loss: 0.2517 - acc: 0.9072 - val_loss: 0.3320 - val_acc: 0.8809
Epoch 11/20
42000/42000 [==============================] - 3s 66us/step - loss: 0.2444 - acc: 0.9085 - val_loss: 0.3332 - val_acc: 0.8799
Epoch 12/20
42000/42000 [==============================] - 3s 66us/step - loss: 0.2363 - acc: 0.9132 - val_loss: 0.3533 - val_acc: 0.8743
Epoch 13/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.2266 - acc: 0.9151 - val_loss: 0.3774 - val_acc: 0.8684
Epoch 14/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.2211 - acc: 0.9179 - val_loss: 0.3218 - val_acc: 0.8888
Epoch 15/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.2137 - acc: 0.9207 - val_loss: 0.3293 - val_acc: 0.8876
Epoch 16/20
42000/42000 [==============================] - 3s 66us/step - loss: 0.2063 - acc: 0.9238 - val_loss: 0.3169 - val_acc: 0.8909
Epoch 17/20
42000/42000 [==============================] - 3s 66us/step - loss: 0.2012 - acc: 0.9254 - val_loss: 0.3309 - val_acc: 0.8857
Epoch 18/20
42000/42000 [==============================] - 3s 66us/step - loss: 0.1979 - acc: 0.9265 - val_loss: 0.3280 - val_acc: 0.8913
Epoch 19/20
42000/42000 [==============================] - 3s 66us/step - loss: 0.1884 - acc: 0.9304 - val_loss: 0.3362 - val_acc: 0.8849
Epoch 20/20
42000/42000 [==============================] - 3s 66us/step - loss: 0.1838 - acc: 0.9316 - val_loss: 0.3301 - val_acc: 0.8886
```





```
60000/60000 [==============================] - 2s 37us/step
10000/10000 [==============================] - 0s 37us/step
Test accuracy: 0.8808
```

■ **Comment:**

- As epoch grows, there may be fluctuation of training accuracy and validation accuracy between the epoch, but overall the general training accuracy and validation accuracy are increasing. By contrast, the validation accuracy is more volatile than the training accuracy.
- As epoch grows, there may be fluctuation of training loss and validation loss between the epoch, but overall the general training loss and validation loss are decreasing. By contrast, the validation loss is more volatile than the training loss.