Erin Howland Week 6 Assignment 6 17 July 2022

In [1]:
```
# import libraries
import cv2
import numpy as np
from matplotlib import pyplot as plt

from keras.datasets import mnist, cifar10
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.utils.np_utils import to_categorical
from keras import backend as K
from keras import models

from sklearn.metrics import confusion_matrix
```

6.1 MNIST

In [2]:
```
# color channel
K.set_image_data_format('channels_last')

# set random seed
np.random.seed(0)

# set image info
channels = 1
height = 28
width = 28
```

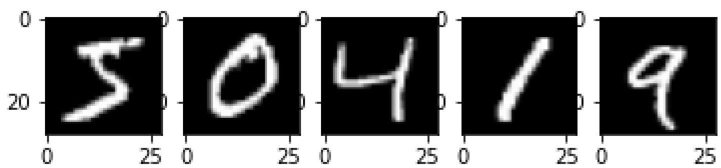
In [3]:
```
# load MNIST data
(data_train, target_train), (data_test, target_test) = mnist.load_data()

# reshape training and test data
data_train = data_train.reshape(data_train.shape[0], height, width,
channels)
data_test = data_test.reshape(data_test.shape[0], height, width, channels)

# view shape of training and test data
```

```
print('data_train: ', data_train.shape)
print('data_test: ', data_test.shape)
```

```
data_train:  (60000, 28, 28, 1)
data_test:  (10000, 28, 28, 1)
```

In [4]:

```
# subplots for first 5 figures of training data
fig, axes = plt.subplots(nrows = 1, ncols = 5)

for idx, ax in enumerate(axes):
    image = data_train[idx]
    ax.imshow(image, cmap = 'gray')
plt.show()
```



In [5]:

```
# ensure training array labels match images
target_train[0:5]
```

Out[5]:

```
array([5, 0, 4, 1, 9], dtype=uint8)
```

In [6]:

```
# rescale pixel intensity
features_train = data_train/255
features_test = data_test/255
```

In [7]:

```
# one hot encoding
target_train = to_categorical(target_train)
target_test = to_categorical(target_test)

number_of_classes = target_test.shape[1]
```

In [8]:

```
# view shapes
print('target_train: ', target_train.shape)
print('target_test: ', target_test.shape)
```

```
target_train:  (60000, 10)
target_test:  (10000, 10)
```

In [9]:

```
# build cnn (20.15)
cnn = Sequential()
```

```python

# convolutional layer
cnn.add(Conv2D(filters = 64,
               kernel_size = (5, 5),
               input_shape = (height, width, channels),
               activation = 'relu'))

# max pooling layer
cnn.add(MaxPooling2D(pool_size = (2,2)))

# dropout layer
cnn.add(Dropout(0.5))

# flatten input
cnn.add(Flatten())

# connected layer with relu activation
cnn.add(Dense(128, activation = 'relu'))

# connected layer with softmax activation
cnn.add(Dense(number_of_classes, activation = 'softmax'))
```

In [10]:

```python
# compile network with cross entropy, root mean square, and accuracy metrics
cnn.compile(loss = 'categorical_crossentropy',
            optimizer = 'rmsprop',
            metrics = ['accuracy'])
```

In [11]:

```python
print(features_train.shape)
print(target_train.shape)
```

```
(60000, 28, 28, 1)
(60000, 10)
```

In [12]:

```python
# train network: print description (verbose)
history = cnn.fit(features_train,
                  target_train,
                  epochs = 5,
                  verbose = 1,
```

```python
                    batch_size = 1000,
                    validation_data = (features_test, target_test))
```

```
Epoch 1/5
60/60 [==============================] - 23s 375ms/step - loss: 0.4923 - accuracy: 0.848
3 - val_loss: 0.1573 - val_accuracy: 0.9560
Epoch 2/5
60/60 [==============================] - 22s 367ms/step - loss: 0.1503 - accuracy: 0.955
4 - val_loss: 0.0890 - val_accuracy: 0.9726
Epoch 3/5
60/60 [==============================] - 21s 349ms/step - loss: 0.0927 - accuracy: 0.972
6 - val_loss: 0.0745 - val_accuracy: 0.9759
Epoch 4/5
60/60 [==============================] - 21s 352ms/step - loss: 0.0655 - accuracy: 0.980
0 - val_loss: 0.0477 - val_accuracy: 0.9848
Epoch 5/5
60/60 [==============================] - 21s 350ms/step - loss: 0.0530 - accuracy: 0.983
9 - val_loss: 0.0438 - val_accuracy: 0.9857
```

In [13]:

```python
# model summary
cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 24, 24, 64)        1664      
                                                                 
 max_pooling2d (MaxPooling2D  (None, 12, 12, 64)       0         
 )                                                               
                                                                 
 dropout (Dropout)           (None, 12, 12, 64)        0         
                                                                 
 flatten (Flatten)           (None, 9216)              0         
                                                                 
 dense (Dense)               (None, 128)               1179776   
                                                                 
 dense_1 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 1,182,730
Trainable params: 1,182,730
Non-trainable params: 0
_________________________________________________________________
```

In [14]:

```python
# look at predictions
predict = cnn.predict(features_test)
predict
```

Out[14]:

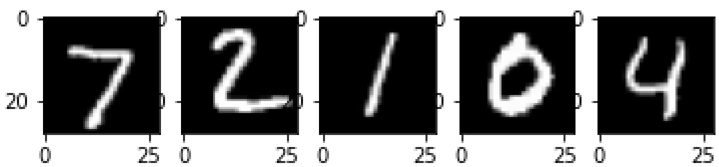
```
array([[1.2988026e-07, 3.3542483e-07, 2.8180423e-06, ..., 9.9998319e-01,
        6.0979556e-07, 5.4090287e-06],
       [2.7343422e-06, 1.3236875e-03, 9.9864119e-01, ..., 5.5630784e-09,
        7.5260232e-06, 1.0937643e-10],
       [5.3205354e-06, 9.9970394e-01, 1.2173172e-05, ..., 1.6088453e-04,
        5.7783702e-05, 4.1878673e-07],
       ...,
       [3.8043582e-09, 1.0894094e-06, 1.3762605e-09, ..., 1.3904720e-05,
        1.1283766e-05, 2.6525971e-05],
       [1.6635209e-08, 1.0133297e-07, 1.9055432e-10, ..., 4.6930822e-08,
```

```
        1.9396354e-04, 1.4946359e-08],
       [2.3958421e-08, 2.8845265e-10, 1.2017881e-08, ..., 1.6367237e-11,
        1.3174086e-08, 7.5311840e-11]], dtype=float32)
```

In [15]:
```python
# make predictions readable
predict = np.argmax(predict, axis = 1)
predict
```

Out[15]:
```
array([7, 2, 1, ..., 4, 5, 6], dtype=int64)
```

In [16]:
```python
# show predicted images
fig, axes = plt.subplots(nrows = 1, ncols = 5)

for idx, ax in enumerate(axes):
    image = data_test[idx]
    ax.imshow(image, cmap = 'gray')
plt.show()
```



In [17]:
```python
# confusion matrix
matrix = confusion_matrix(target_test.argmax(axis = 1), predict)
matrix
```

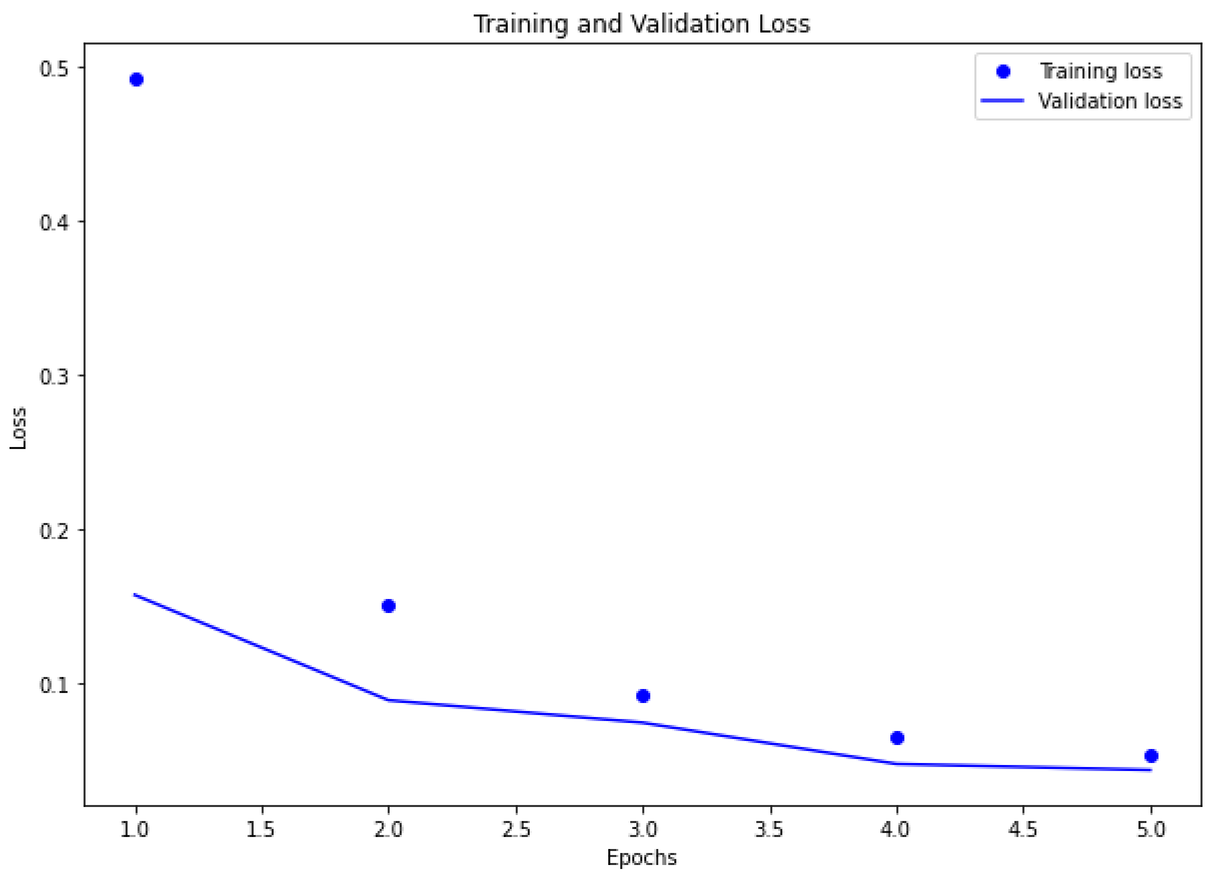
Out[17]:
```
array([[ 975,    0,    0,    0,    0,    0,    2,    1,    2,    0],
       [   0, 1132,    0,    1,    0,    0,    1,    0,    1,    0],
       [   3,    8,  998,    4,    1,    0,    1,   11,    5,    1],
       [   0,    0,    2,  999,    0,    1,    0,    6,    2,    0],
       [   0,    0,    1,    0,  973,    0,    1,    0,    3,    4],
       [   2,    0,    0,    5,    0,  882,    2,    0,    1,    0],
       [   7,    2,    0,    0,    1,    3,  943,    0,    2,    0],
       [   0,    1,    4,    2,    0,    0,    0, 1020,    1,    0],
       [   5,    0,    2,    1,    1,    0,    0,    3,  961,    1],
       [   3,    7,    0,    2,    6,    3,    0,   11,    3,  974]],
      dtype=int64)
```

In [18]:
```python
history_dict = history.history
history_dict.keys()
```

Out[18]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [19]:
```python
# plot training and validation loss
plt.figure(figsize=(10,7))
```

```
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

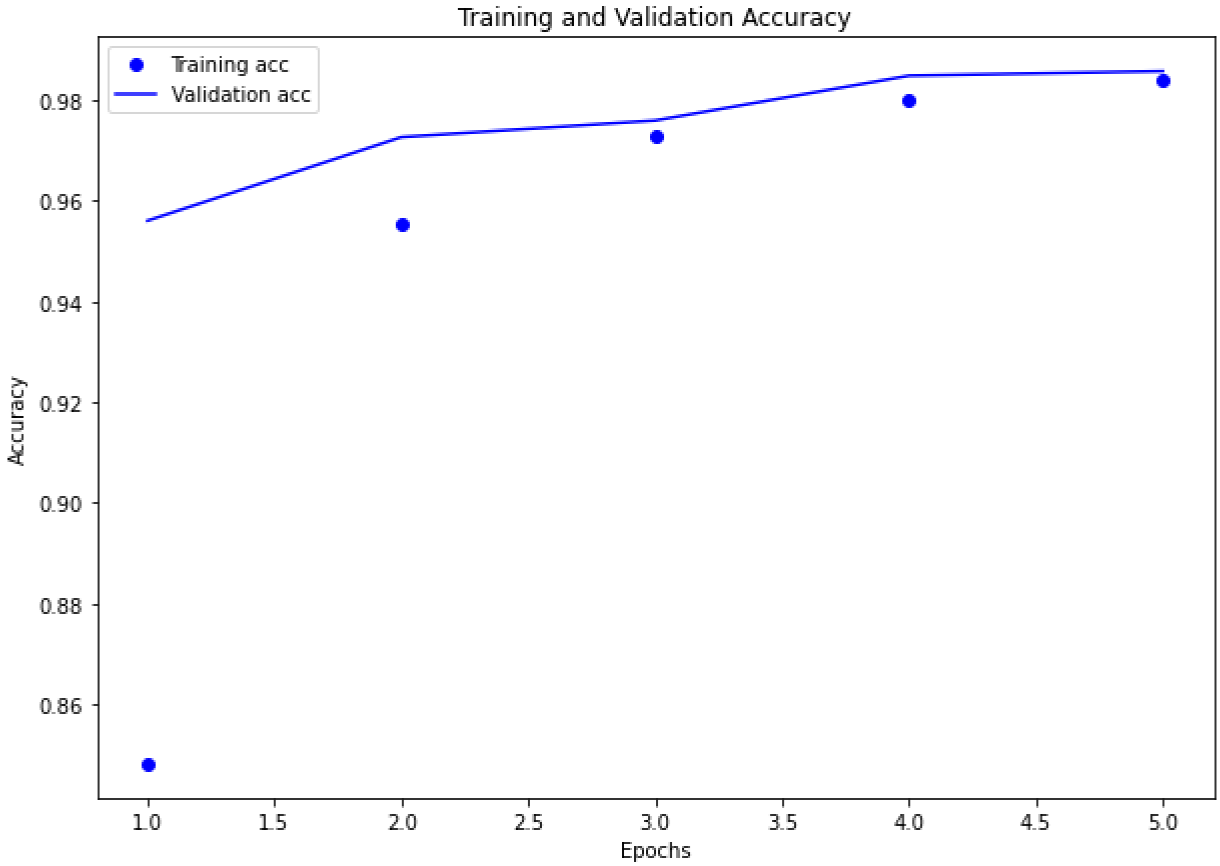


In [20]:

```
plt.clf()

plt.figure(figsize=(10,7))

acc = history_dict["accuracy"]
val_acc = history_dict["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
```

```
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



6.2: CIFAR10 documentation: https://keras.io/api/datasets/cifar10/

In [21]:

```
# load and split data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

In [22]:

```
# show sizes
print('x_train shape: ', x_train.shape)
print('y_train shape: ', y_train.shape)
print('x_test shape: ', x_test.shape)
print('x_train shape: ', x_train.shape)
```

```
x_train shape:  (50000, 32, 32, 3)
y_train shape:  (50000, 1)
x_test shape:  (10000, 32, 32, 3)
x_train shape:  (50000, 32, 32, 3)
```

In [23]:

```
cifar10_classes = ['airplane',
                   'automobile',
                   'bird',
```

```
                    'cat',
                    'deer',
                    'dog',
                    'frog',
                    'horse',
                    'ship',
                    'truck']
```

In [24]:

```
# split data into train and validation

# scale the data
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# one-hot encoding
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

In [25]:

```
# view sizes
print('x-train scaled shape: ', x_train.shape)
print('x-test scaled shape: ', x_test.shape)

print('y_train ohe shape: ', y_train.shape)
print('y_test ohe shape: ', y_test.shape)

print('partial x_train shape: ', partial_x_train.shape)
print('partial y_train shape: ', partial_y_train.shape)
```

```
x-train scaled shape:  (50000, 32, 32, 3)
x-test scaled shape:  (10000, 32, 32, 3)
y_train ohe shape:  (50000, 10)
y_test ohe shape:  (10000, 10)
partial x_train shape:  (40000, 32, 32, 3)
partial y_train shape:  (40000, 10)
```

In [26]:

```
# build model
```

```python
model = models.Sequential()

model.add(Conv2D(32, (3, 3),
                        activation='relu',
                        padding='same',
                        input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3),
                        activation='relu',
                        padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3),
                        activation='relu',
                        padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3),
                        activation='relu',
                        padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='sigmoid'))
```

In [27]:

```python
# model summary
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_1 (Conv2D)           (None, 32, 32, 32)        896       
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496     
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 8, 8, 64)         0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (None, 8, 8, 128)         73856     
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 4, 4, 128)        0         
 2D)                                                             
                                                                 
 conv2d_4 (Conv2D)           (None, 4, 4, 128)         147584    
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 2, 2, 128)        0         
```

```
 2D)

 flatten_1 (Flatten)         (None, 512)               0

 dense_2 (Dense)             (None, 512)               262656

 dense_3 (Dense)             (None, 10)                5130

=================================================================
Total params: 508,618
Trainable params: 508,618
Non-trainable params: 0
_________________________________________________________________
```

In [28]:
```python
# compile model
from tensorflow.keras import optimizers

model.compile(loss = 'binary_crossentropy',
              optimizer = optimizers.RMSprop(learning_rate = 1e-4),
              metrics = ['acc'])
```

In [29]:
```python
history = model.fit(partial_x_train, partial_y_train,
                    epochs=30,
                    batch_size=64,
                    validation_data = (x_val,y_val),
                    validation_steps=50)
```

```
Epoch 1/30
625/625 [==============================] - 39s 62ms/step - loss: 0.2969 - acc: 0.2765 -
val_loss: 0.2666 - val_acc: 0.3559
Epoch 2/30
625/625 [==============================] - 39s 62ms/step - loss: 0.2551 - acc: 0.3989 -
val_loss: 0.2400 - val_acc: 0.4325
Epoch 3/30
625/625 [==============================] - 38s 61ms/step - loss: 0.2386 - acc: 0.4406 -
val_loss: 0.2305 - val_acc: 0.4619
Epoch 4/30
625/625 [==============================] - 38s 60ms/step - loss: 0.2269 - acc: 0.4771 -
val_loss: 0.2197 - val_acc: 0.4953
Epoch 5/30
625/625 [==============================] - 38s 61ms/step - loss: 0.2172 - acc: 0.5049 -
val_loss: 0.2176 - val_acc: 0.5025
Epoch 6/30
625/625 [==============================] - 42s 67ms/step - loss: 0.2084 - acc: 0.5312 -
val_loss: 0.2053 - val_acc: 0.5444
Epoch 7/30
625/625 [==============================] - 39s 63ms/step - loss: 0.2006 - acc: 0.5534 -
val_loss: 0.1932 - val_acc: 0.5706
Epoch 8/30
625/625 [==============================] - 40s 64ms/step - loss: 0.1935 - acc: 0.5727 -
val_loss: 0.1868 - val_acc: 0.5919
Epoch 9/30
625/625 [==============================] - 38s 61ms/step - loss: 0.1870 - acc: 0.5916 -
val_loss: 0.1845 - val_acc: 0.5969
Epoch 10/30
625/625 [==============================] - 38s 61ms/step - loss: 0.1808 - acc: 0.6086 -
```

```
val_loss: 0.1815 - val_acc: 0.5981
Epoch 11/30
625/625 [==============================] - 38s 61ms/step - loss: 0.1750 - acc: 0.6234 -
val_loss: 0.1781 - val_acc: 0.6106
Epoch 12/30
625/625 [==============================] - 38s 61ms/step - loss: 0.1693 - acc: 0.6358 -
val_loss: 0.1730 - val_acc: 0.6316
Epoch 13/30
625/625 [==============================] - 38s 61ms/step - loss: 0.1645 - acc: 0.6502 -
val_loss: 0.1674 - val_acc: 0.6394
Epoch 14/30
625/625 [==============================] - 38s 61ms/step - loss: 0.1596 - acc: 0.6623 -
val_loss: 0.1648 - val_acc: 0.6503
Epoch 15/30
625/625 [==============================] - 38s 61ms/step - loss: 0.1551 - acc: 0.6762 -
val_loss: 0.1594 - val_acc: 0.6653
Epoch 16/30
625/625 [==============================] - 38s 61ms/step - loss: 0.1506 - acc: 0.6856 -
val_loss: 0.1640 - val_acc: 0.6494
Epoch 17/30
625/625 [==============================] - 38s 61ms/step - loss: 0.1463 - acc: 0.6964 -
val_loss: 0.1528 - val_acc: 0.6791
Epoch 18/30
625/625 [==============================] - 38s 61ms/step - loss: 0.1423 - acc: 0.7078 -
val_loss: 0.1579 - val_acc: 0.6616
Epoch 19/30
625/625 [==============================] - 38s 61ms/step - loss: 0.1385 - acc: 0.7167 -
val_loss: 0.1493 - val_acc: 0.6909
Epoch 20/30
625/625 [==============================] - 38s 61ms/step - loss: 0.1349 - acc: 0.7258 -
val_loss: 0.1470 - val_acc: 0.6944
Epoch 21/30
625/625 [==============================] - 39s 62ms/step - loss: 0.1313 - acc: 0.7341 -
val_loss: 0.1510 - val_acc: 0.6828
Epoch 22/30
625/625 [==============================] - 39s 62ms/step - loss: 0.1281 - acc: 0.7411 -
val_loss: 0.1447 - val_acc: 0.7009
Epoch 23/30
625/625 [==============================] - 39s 62ms/step - loss: 0.1244 - acc: 0.7481 -
val_loss: 0.1418 - val_acc: 0.7016
Epoch 24/30
625/625 [==============================] - 39s 62ms/step - loss: 0.1211 - acc: 0.7577 -
val_loss: 0.1427 - val_acc: 0.7078
Epoch 25/30
625/625 [==============================] - 39s 62ms/step - loss: 0.1183 - acc: 0.7637 -
val_loss: 0.1380 - val_acc: 0.7225
Epoch 26/30
625/625 [==============================] - 39s 62ms/step - loss: 0.1147 - acc: 0.7722 -
val_loss: 0.1421 - val_acc: 0.7066
Epoch 27/30
625/625 [==============================] - 39s 63ms/step - loss: 0.1122 - acc: 0.7781 -
val_loss: 0.1513 - val_acc: 0.6894
Epoch 28/30
625/625 [==============================] - 39s 63ms/step - loss: 0.1091 - acc: 0.7857 -
val_loss: 0.1401 - val_acc: 0.7094
Epoch 29/30
625/625 [==============================] - 39s 63ms/step - loss: 0.1063 - acc: 0.7932 -
val_loss: 0.1395 - val_acc: 0.7219
Epoch 30/30
625/625 [==============================] - 40s 63ms/step - loss: 0.1033 - acc: 0.8009 -
val_loss: 0.1419 - val_acc: 0.7219
```

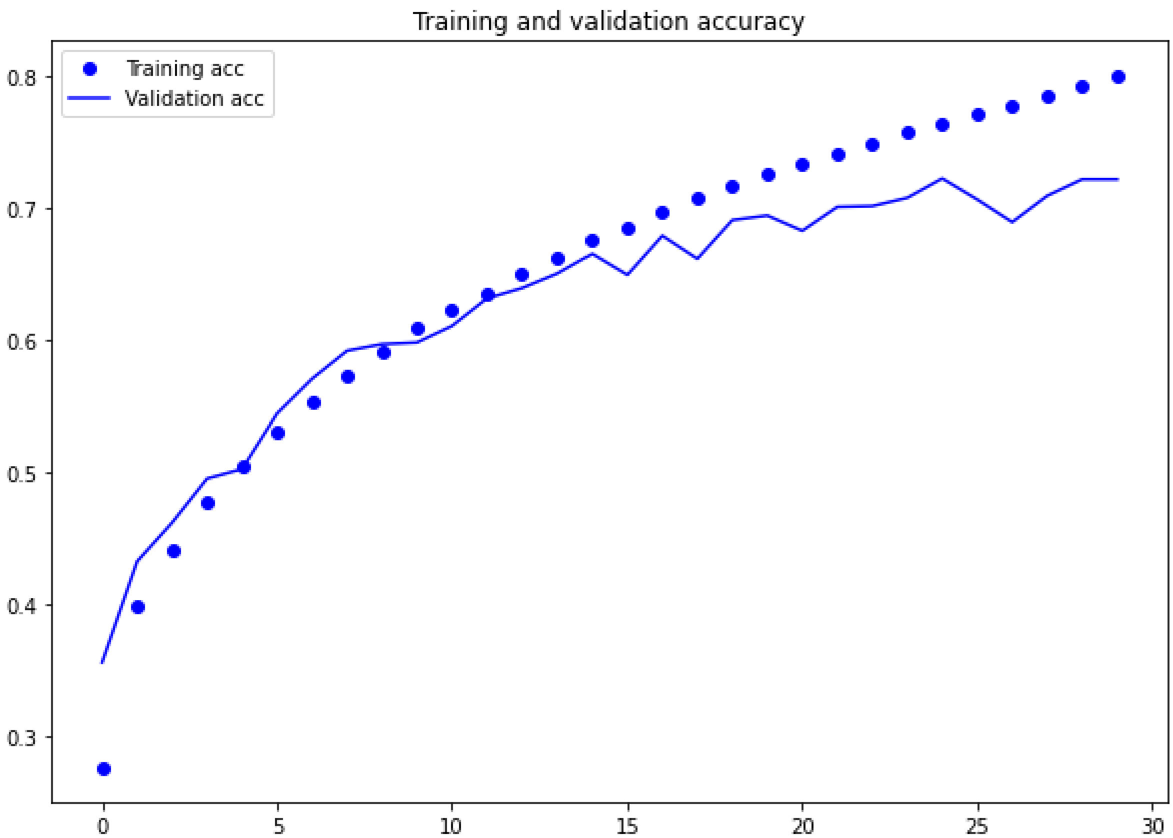
In [30]:

```
# save model
```

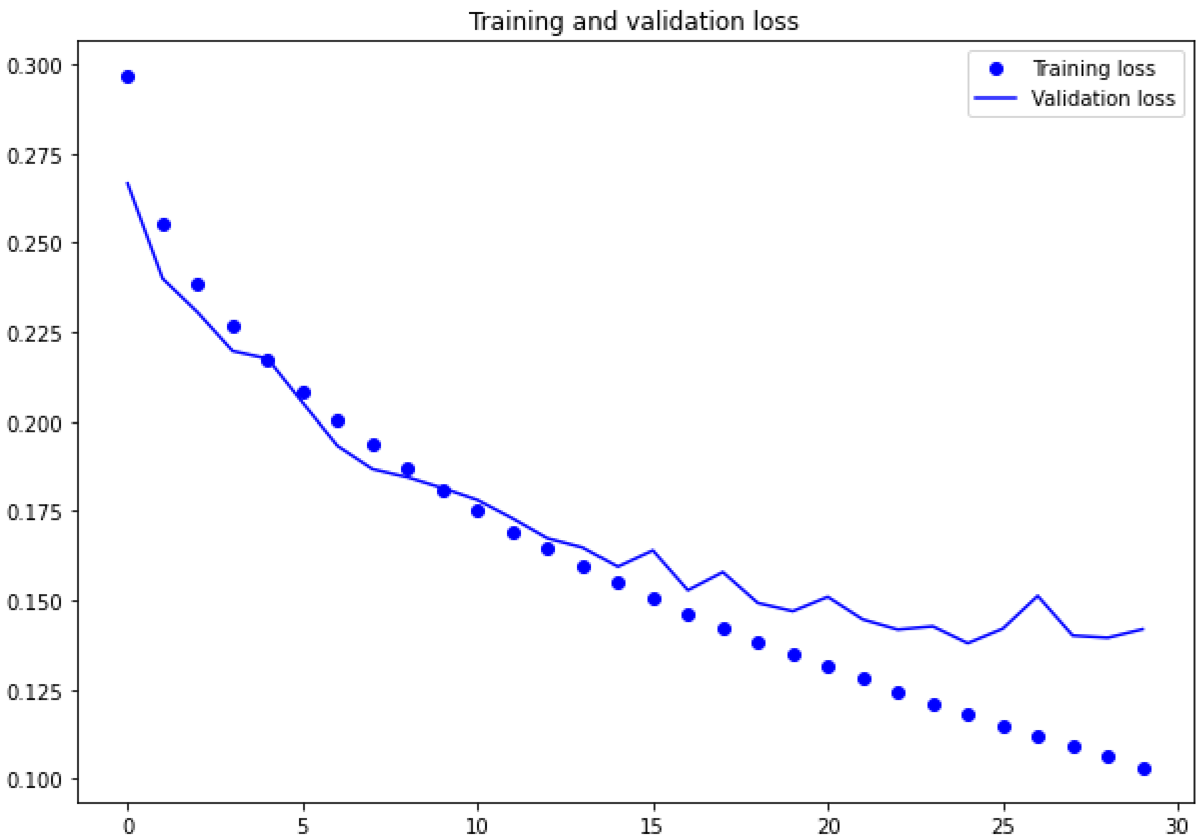
```python
model.save('cnn_classifier_1.h5')
```

In [31]:

```python
# plot training and validation accuracy
plt.figure(figsize=(10,7))

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()



plt.figure()
plt.figure(figsize=(10,7))
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```




```
<Figure size 432x288 with 0 Axes>
```



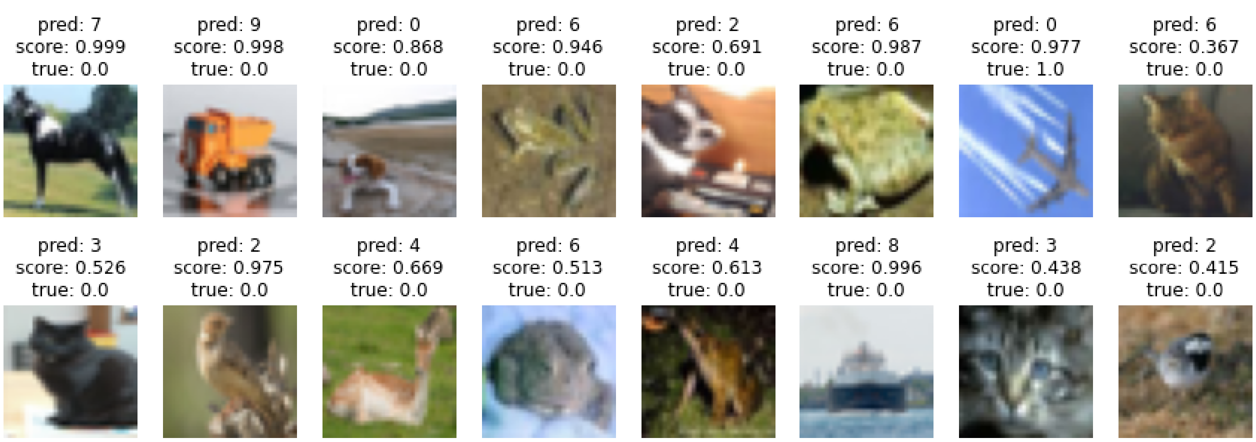
In [32]:

```
results = model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 2s 8ms/step - loss: 0.1464 - acc: 0.7045
```

In [33]:
```
# make predictions
y_pred_test = model.predict(x_test)
y_pred_test_classes = np.argmax(y_pred_test, axis=1)
y_pred_test_max_probas = np.max(y_pred_test, axis=1)
```

In [34]:
```
# show predictions
cols = 8
rows = 2
fig = plt.figure(figsize=(2 * cols - 1, 3 * rows - 1))
for i in range(cols):
    for j in range(rows):
        random_index = np.random.randint(0, len(y_test))
        ax = fig.add_subplot(rows, cols, i * rows + j + 1)
        ax.grid('off')
        ax.axis('off')
        ax.imshow(x_test[random_index, :])
        pred_label = y_pred_test_classes[random_index]
        pred_proba = y_pred_test_max_probas[random_index]
        true_label = y_test[random_index, 0]
        ax.set_title("pred: {}\nscore: {:.3}\ntrue: {}".format(
               pred_label, pred_proba, true_label
        ))
plt.show()
```



6.2

In [35]:
```
# load data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

In [36]:
```
# scale the data
```

```
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# transform target variable into one-hot encoding
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

In [37]:

```
# build the model
model = models.Sequential()
model.add(Conv2D(32, (3, 3),
                        activation='relu',
                        padding='same',
                        input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3),
                        activation='relu',
                        padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3),
                        activation='relu',
                        padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3),
                        activation='relu',
                        padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
```

```
                optimizer = optimizers.RMSprop(learning_rate=1e-4),
                metrics = ['acc'])
```

In [38]:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator


batch_size = 32
data_generator = ImageDataGenerator(width_shift_range=0.1,
height_shift_range=0.1, horizontal_flip=True)
train_generator = data_generator.flow(partial_x_train, partial_y_train,
batch_size)
```

In [39]:

```
#  train the model using data augmentation and dropout:
history = model.fit(train_generator,
                    steps_per_epoch=100,
                    epochs=30,
                    validation_data=(x_test, y_test))
```

```
Epoch 1/30
100/100 [==============================] - 7s 60ms/step - loss: 2.2985 - acc: 0.1191 - v
al_loss: 2.2834 - val_acc: 0.1507
Epoch 2/30
100/100 [==============================] - 6s 56ms/step - loss: 2.2591 - acc: 0.1484 - v
al_loss: 2.1915 - val_acc: 0.2339
Epoch 3/30
100/100 [==============================] - 6s 56ms/step - loss: 2.1358 - acc: 0.2019 - v
al_loss: 2.0563 - val_acc: 0.2506
Epoch 4/30
100/100 [==============================] - 6s 57ms/step - loss: 2.0601 - acc: 0.2262 - v
al_loss: 2.0441 - val_acc: 0.2490
Epoch 5/30
100/100 [==============================] - 6s 56ms/step - loss: 1.9920 - acc: 0.2650 - v
al_loss: 1.9223 - val_acc: 0.3049
Epoch 6/30
100/100 [==============================] - 6s 56ms/step - loss: 1.9489 - acc: 0.2837 - v
al_loss: 1.8340 - val_acc: 0.3415
Epoch 7/30
100/100 [==============================] - 6s 56ms/step - loss: 1.9033 - acc: 0.2891 - v
al_loss: 1.8829 - val_acc: 0.3152
Epoch 8/30
100/100 [==============================] - 6s 56ms/step - loss: 1.8440 - acc: 0.3203 - v
al_loss: 1.7753 - val_acc: 0.3720
Epoch 9/30
100/100 [==============================] - 6s 56ms/step - loss: 1.8355 - acc: 0.3153 - v
al_loss: 1.7316 - val_acc: 0.3690
Epoch 10/30
100/100 [==============================] - 6s 55ms/step - loss: 1.8028 - acc: 0.3397 - v
al_loss: 1.7959 - val_acc: 0.3627
Epoch 11/30
100/100 [==============================] - 6s 63ms/step - loss: 1.7923 - acc: 0.3313 - v
al_loss: 1.6993 - val_acc: 0.3911
Epoch 12/30
100/100 [==============================] - 6s 61ms/step - loss: 1.7823 - acc: 0.3516 - v
al_loss: 1.7007 - val_acc: 0.3888
Epoch 13/30
```

```
100/100 [==============================] - 6s 57ms/step - loss: 1.7340 - acc: 0.3697 - v
al_loss: 1.6718 - val_acc: 0.3931
Epoch 14/30
100/100 [==============================] - 6s 56ms/step - loss: 1.7275 - acc: 0.3631 - v
al_loss: 1.6294 - val_acc: 0.4090
Epoch 15/30
100/100 [==============================] - 6s 58ms/step - loss: 1.7008 - acc: 0.3731 - v
al_loss: 1.6064 - val_acc: 0.4121
Epoch 16/30
100/100 [==============================] - 6s 57ms/step - loss: 1.6689 - acc: 0.3887 - v
al_loss: 1.6417 - val_acc: 0.3916
Epoch 17/30
100/100 [==============================] - 6s 57ms/step - loss: 1.6760 - acc: 0.3841 - v
al_loss: 1.5547 - val_acc: 0.4268
Epoch 18/30
100/100 [==============================] - 6s 58ms/step - loss: 1.6656 - acc: 0.3900 - v
al_loss: 1.5871 - val_acc: 0.4155
Epoch 19/30
100/100 [==============================] - 6s 57ms/step - loss: 1.6346 - acc: 0.4034 - v
al_loss: 1.6403 - val_acc: 0.4174
Epoch 20/30
100/100 [==============================] - 6s 58ms/step - loss: 1.6501 - acc: 0.3794 - v
al_loss: 1.5258 - val_acc: 0.4345
Epoch 21/30
100/100 [==============================] - 6s 57ms/step - loss: 1.6033 - acc: 0.4116 - v
al_loss: 1.5748 - val_acc: 0.4188
Epoch 22/30
100/100 [==============================] - 6s 57ms/step - loss: 1.6249 - acc: 0.4081 - v
al_loss: 1.5282 - val_acc: 0.4292
Epoch 23/30
100/100 [==============================] - 6s 58ms/step - loss: 1.5845 - acc: 0.4291 - v
al_loss: 1.5764 - val_acc: 0.4168
Epoch 24/30
100/100 [==============================] - 6s 57ms/step - loss: 1.5848 - acc: 0.4250 - v
al_loss: 1.4659 - val_acc: 0.4646
Epoch 25/30
100/100 [==============================] - 6s 57ms/step - loss: 1.5991 - acc: 0.4212 - v
al_loss: 1.4751 - val_acc: 0.4556
Epoch 26/30
100/100 [==============================] - 6s 58ms/step - loss: 1.5692 - acc: 0.4284 - v
al_loss: 1.5273 - val_acc: 0.4309
Epoch 27/30
100/100 [==============================] - 6s 57ms/step - loss: 1.6083 - acc: 0.3953 - v
al_loss: 1.4969 - val_acc: 0.4598
Epoch 28/30
100/100 [==============================] - 6s 60ms/step - loss: 1.5771 - acc: 0.4212 - v
al_loss: 1.4582 - val_acc: 0.4700
Epoch 29/30
100/100 [==============================] - 6s 58ms/step - loss: 1.5754 - acc: 0.4325 - v
al_loss: 1.5632 - val_acc: 0.4324
Epoch 30/30
100/100 [==============================] - 6s 56ms/step - loss: 1.5624 - acc: 0.4325 - v
al_loss: 1.6994 - val_acc: 0.3989
```

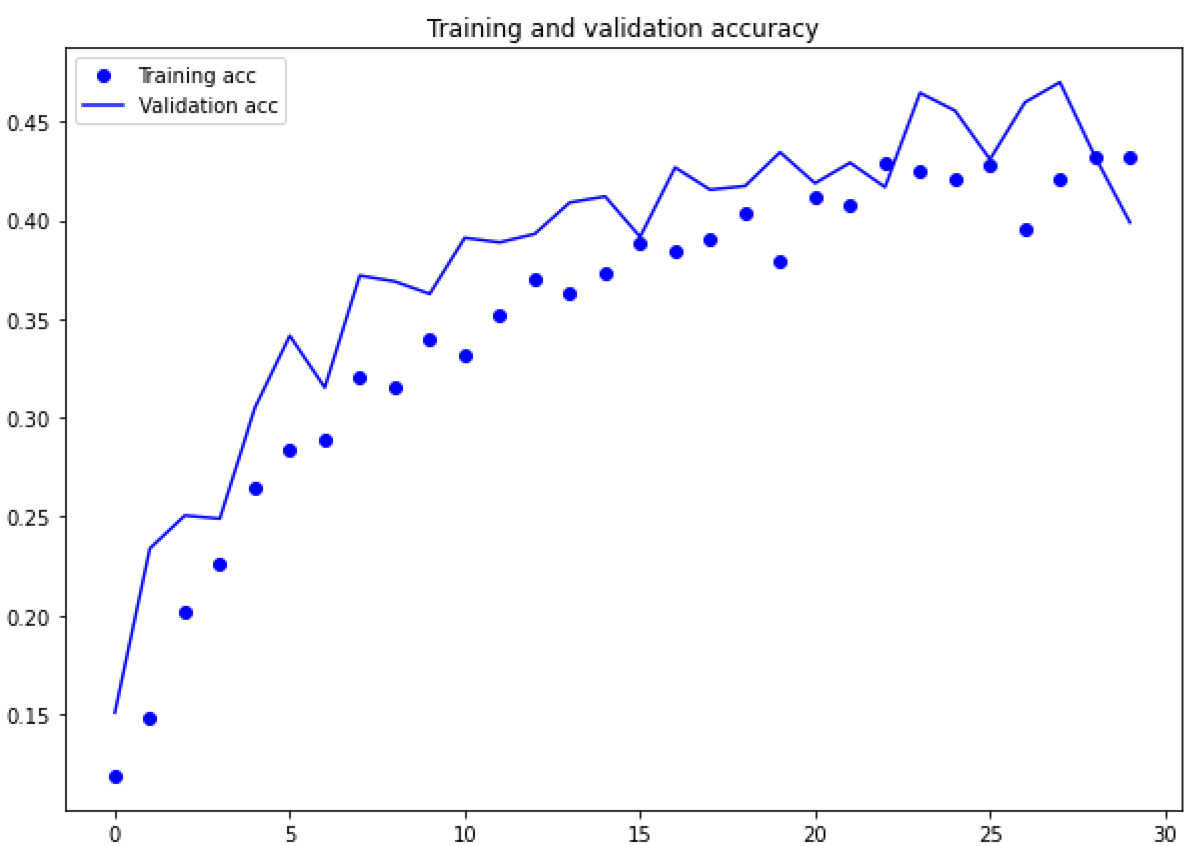
In [40]:

```python
#  save the model
model.save('cnn_classifier_2.h5')
```

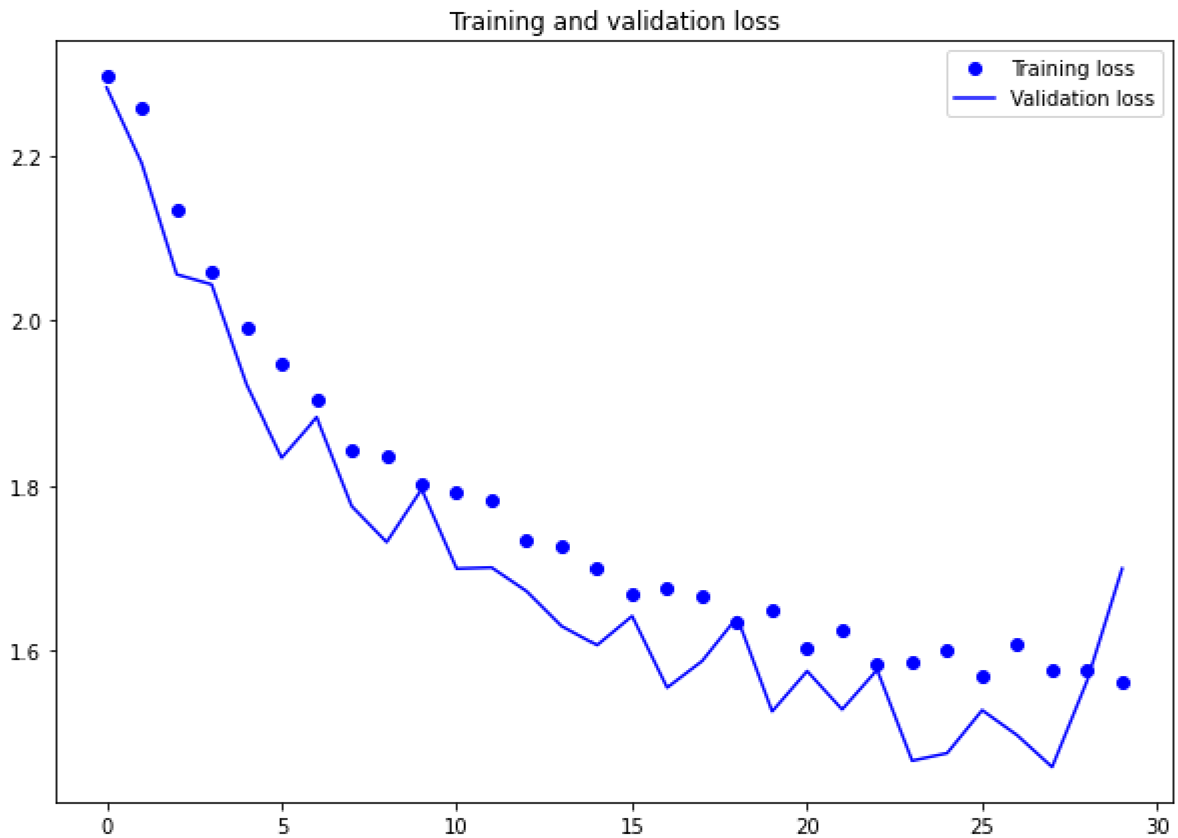
In [41]:

```python
# plot the training vs validation - accuracy and loss


plt.figure(figsize=(10,7))
```

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()
plt.figure(figsize=(10,7))
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```




```
<Figure size 432x288 with 0 Axes>
```



In [42]:

```
# predicting test data
y_pred_test = model.predict(x_test)
y_pred_test_classes = np.argmax(y_pred_test, axis=1)
y_pred_test_max_probas = np.max(y_pred_test, axis=1)
```

In [44]:

```
# display the predictions
cols = 8
rows = 2
fig = plt.figure(figsize=(2 * cols - 1, 3 * rows - 1))
for i in range(cols):
    for j in range(rows):
        random_index = np.random.randint(0, len(y_test))
        ax = fig.add_subplot(rows, cols, i * rows + j + 1)
        ax.grid('off')
        ax.axis('off')
        ax.imshow(x_test[random_index, :])
        pred_label = y_pred_test_classes[random_index]
        pred_proba = y_pred_test_max_probas[random_index]
        true_label = y_test[random_index, 0]
        ax.set_title("pred: {}\nscore: {:.3}\ntrue: {}".format(
               pred_label, pred_proba, true_label
```

```
        ))
plt.show()
```

6.3 documentation: https://keras.io/api/applications/#classify-imagenet-classes-with-resnet50

In [45]:
```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input,
decode_predictions
```

In [ ]:
```
```