CNN Image Classification Laboration

Images used in this laboration are from CIFAR 10 (https://en.wikipedia.org/wiki/CIFAR-10). The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class. Your task is to make a classifier, using a convolutional neural network, that can correctly classify each image into the correct class.

You need to answer all questions in this notebook.

Part 1: What is a convolution

To understand a bit more about convolutions, we will first test the convolution function in scipy using a number of classical filters.

Convolve the image with Gaussian filter, a Sobel X filter, and a Sobel Y filter, using the function 'convolve2d' in 'signal' from scipy.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.convolve2d.html

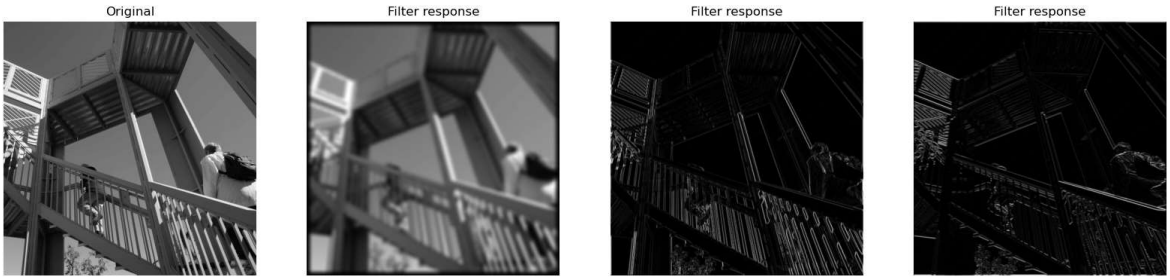
In a CNN, many filters are applied in each layer, and the filter coefficients are learned through back propagation (which is in contrast to traditional image processing, where the filters are designed by an expert).

In [ ]:
```
# This cell is finished

from scipy import signal
import numpy as np

# Get a test image
from scipy import misc
image = misc.ascent()

# Define a help function for creating a Gaussian filter
def matlab_style_gauss2D(shape=(3,3),sigma=0.5):
    """
    2D gaussian mask - should give the same result as MATLAB's
    fspecial('gaussian',[shape],[sigma])
    """
    m,n = [(ss-1.)/2. for ss in shape]
    y,x = np.ogrid[-m:m+1,-n:n+1]
    h = np.exp( -(x*x + y*y) / (2.*sigma*sigma) )
    h[ h < np.finfo(h.dtype).eps*h.max() ] = 0
    sumh = h.sum()
    if sumh != 0:
        h /= sumh
    return h

# Create Gaussian filter with certain size and standard deviation
gaussFilter = matlab_style_gauss2D((15,15),4)
```

```
# Define filter kernels for SobelX and SobelY
sobelX = np.array([[ 1, 0,  -1],
                   [2, 0, -2],
                   [1, 0, -1]])

sobelY = np.array([[ 1, 2,  1],
                   [0, 0, 0],
                   [-1, -2, -1]])
```

In [ ]:
```
# Perform convolution using the function 'convolve2d' for the different filters
filterResponseGauss = signal.convolve2d(image,gaussFilter)

filterResponseSobelX = signal.convolve2d(image,sobelX)
filterResponseSobelY = signal.convolve2d(image,sobelY)
```

In [ ]:
```
import matplotlib.pyplot as plt

# Show filter responses
fig, (ax_orig, ax_filt1, ax_filt2, ax_filt3) = plt.subplots(1, 4, figsize=(20, 6
ax_orig.imshow(image, cmap='gray')
ax_orig.set_title('Original')
ax_orig.set_axis_off()
ax_filt1.imshow(np.absolute(filterResponseGauss), cmap='gray')
ax_filt1.set_title('Filter response')
ax_filt1.set_axis_off()
ax_filt2.imshow(np.absolute(filterResponseSobelX), cmap='gray')
ax_filt2.set_title('Filter response')
ax_filt2.set_axis_off()
ax_filt3.imshow(np.absolute(filterResponseSobelY), cmap='gray')
ax_filt3.set_title('Filter response')
ax_filt3.set_axis_off()
```



Part 2: Understanding convolutions

Question 1: What do the 3 different filters (Gaussian, SobelX, SobelY) do to the original image?

- Gaussian: blur the image
- SobelX : find edges horizontally
- SobelY : find edges vertically

Question 2: What is the size of the original image? How many channels does it have? How many channels does a color image normally have?

- size of the original image is 512x512
- it's grayscale image thus it has 1 channel

- a color image normally have 3 channels: RGB

Question 3: What is the size of the different filters?

- Gaussian: 15x15
- SobelX : 3x3
- SobelY : 3x3

Question 4: What is the size of the filter response if mode 'same' is used for the convolution ?

- mode = 'same':The output is the same size as in1, centered with respect to the 'full' output.
- the size of output will be 512x512, the same as input image.

Question 5: What is the size of the filter response if mode 'valid' is used for the convolution? How does the size of the valid filter response depend on the size of the filter?

- mode = 'valid':The output consists only of those elements that do not rely on the zero-padding. In 'valid' mode, either in1 or in2 must be at least as large as the other in every dimension.
- the size will be (512-filtersizeX) x (512-filtersizeY)

Question 6: Why are 'valid' convolutions a problem for CNNs with many layers?

- it will reduce the dimensions of input by and by when we have many layers. We need to bear it in mind esspecially when we dont want such characteristic.

In [ ]:
```
# Your code for checking sizes of image and filter responses
print(image.shape)
print(signal.convolve2d(image,gaussFilter,mode='same').shape)
print(signal.convolve2d(image,gaussFilter,mode='valid').shape)
```

```
(512, 512)
(512, 512)
(498, 498)
(498, 498)
```

Part 3: Get a graphics card

Skip this part if you run on a CPU (recommended)

Let's make sure that our script can see the graphics card that will be used. The graphics cards will perform all the time consuming convolutions in every training iteration.

In [ ]:
```
import os
import warnings

# Ignore FutureWarning from numpy
warnings.simplefilter(action='ignore', category=FutureWarning)

import keras.backend as K
```

```
import tensorflow as tf

os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID";

# The GPU id to use, usually either "0" or "1";
os.environ["CUDA_VISIBLE_DEVICES"]="0";

# Allow growth of GPU memory, otherwise it will always look like all the memory
physical_devices = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)
```

Part 4: How fast is the graphics card?

Question 7: Why are the filters used for a color image of size 7 x 7 x 3, and not 7 x 7 ?

- there're 3 channels(RGB) for color image besides image sieze 7x7

Question 8: What operation is performed by the 'Conv2D' layer? Is it a standard 2D convolution, as performed by the function signal.convolve2d we just tested?

- in 'Conv2D', it's using crosscorrelation, instead of standard 2D convolution

Question 9: Do you think that a graphics card, compared to the CPU, is equally faster for convolving a batch of 1,000 images, compared to convolving a batch of 3 images? Motivate your answer.

- yes, graphics card is good at compute massive simples tasks parallelly, compared to the CPU, which has more powerful but much less cores than GPU

Part 5: Load data

Time to make a 2D CNN. Load the images and labels from keras.datasets, this cell is already finished.

In [ ]:
```
from keras.datasets import cifar10
import numpy as np

classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship'

# Download CIFAR train and test data
(Xtrain, Ytrain), (Xtest, Ytest) = cifar10.load_data()

print("Training images have size {} and labels have size {} ".format(Xtrain.shap
print("Test images have size {} and labels have size {} \n ".format(Xtest.shape,

# Reduce the number of images for training and testing to 10000 and 2000 respect
# to reduce processing time for this laboration
Xtrain = Xtrain[0:10000]
Ytrain = Ytrain[0:10000]

Xtest = Xtest[0:2000]
Ytest = Ytest[0:2000]

Ytestint = Ytest
```

```
print("Reduced training images have size %s and labels have size %s " % (Xtrain.
print("Reduced test images have size %s and labels have size %s \n" % (Xtest.sha

# Check that we have some training examples from each class
for i in range(10):
    print("Number of training examples for class {} is {}" .format(i,np.sum(Ytra
```

```
Training images have size (50000, 32, 32, 3) and labels have size (50000, 1)
Test images have size (10000, 32, 32, 3) and labels have size (10000, 1)

Reduced training images have size (10000, 32, 32, 3) and labels have size (10000,
1)
Reduced test images have size (2000, 32, 32, 3) and labels have size (2000, 1)

Number of training examples for class 0 is 1005
Number of training examples for class 1 is 974
Number of training examples for class 2 is 1032
Number of training examples for class 3 is 1016
Number of training examples for class 4 is 999
Number of training examples for class 5 is 937
Number of training examples for class 6 is 1030
Number of training examples for class 7 is 1001
Number of training examples for class 8 is 1025
Number of training examples for class 9 is 981
```

Part 6: Plotting

Lets look at some of the training examples, this cell is already finished. You will see different examples every time you run the cell.

In [ ]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(12,4))
for i in range(18):
    idx = np.random.randint(7500)
    label = Ytrain[idx,0]

    plt.subplot(3,6,i+1)
    plt.tight_layout()
    plt.imshow(Xtrain[idx])
    plt.title("Class: {} ({})".format(label, classes[label]))
    plt.axis('off')
plt.show()
```

Part 7: Split data into training, validation and testing

Split your training data into training (Xtrain, Ytrain) and validation (Xval, Yval), so that we have training, validation and test datasets (as in the previous laboration). We use a function in scikit learn. Use 25% of the data for validation.

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

In [ ]:
```
from sklearn.model_selection import train_test_split

# Your code for splitting the dataset
Xtrain, Xval, Ytrain, Yval = train_test_split(Xtrain, Ytrain, test_size=0.3, rar
# Print the size of training data, validation data and test data
for set in (Xtrain, Xval, Ytrain, Yval):
    print(f'{set.shape}')
```

```
(7000, 32, 32, 3)
(3000, 32, 32, 3)
(7000, 1)
(3000, 1)
```

Part 8: Preprocessing of images

Lets perform some preprocessing. The images are stored as uint8, i.e. 8 bit unsigned integers, but need to be converted to 32 bit floats. We also make sure that the range is -1 to 1, instead of 0 - 255. This cell is already finished.

In [ ]:
```
# Convert datatype for Xtrain, Xval, Xtest, to float32
Xtrain = Xtrain.astype('float32')
Xval = Xval.astype('float32')
Xtest = Xtest.astype('float32')

# Change range of pixel values to [-1,1]
Xtrain = Xtrain / 127.5 - 1
Xval = Xval / 127.5 - 1
Xtest = Xtest / 127.5 - 1
```

Part 9: Preprocessing of labels

The labels (Y) need to be converted from e.g. '4' to "hot encoded", i.e. to a vector of type [0, 0, 0, 1, 0, 0, 0, 0, 0, 0] . We use a function in Keras, see https://keras.io/api/utils/python_utils/#to_categorical-function

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

# for set in (Ytrain, Yval, Ytest):
# # Print shapes before converting the labels
#     print('before converting',set.shape)
# # Your code for converting Ytrain, Yval, Ytest to categorical
```

```
#     set = to_categorical(set, num_classes=len(classes))
# # Print shapes after converting the labels
#     print('after converting',set.shape)

# for set in (Ytrain, Yval, Ytest):
#     print('after converting',set.shape)
# # work only within the loop, why?
print(f'Ytrain.shape = {Ytrain.shape}')
print(f'Yval.shape = {Yval.shape}')
print(f'Ytest.shape = {Ytest.shape}')

Ytrain = to_categorical(Ytrain, num_classes=len(classes))
Yval = to_categorical(Yval, num_classes=len(classes))
Ytest = to_categorical(Ytest, num_classes=len(classes))

print('after converting:')

print(f'Ytrain.shape = {Ytrain.shape}')
print(f'Yval.shape = {Yval.shape}')
print(f'Ytest.shape = {Ytest.shape}')
```

```
Ytrain.shape = (7000, 1)
Yval.shape = (3000, 1)
Ytest.shape = (2000, 1)
after converting:
Ytrain.shape = (7000, 10)
Yval.shape = (3000, 10)
Ytest.shape = (2000, 10)
```

Part 10: 2D CNN

Finish this code to create the image classifier, using a 2D CNN. Each convolutional layer will contain 2D convolution, batch normalization and max pooling. After the convolutional layers comes a flatten layer and a number of intermediate dense layers. The convolutional layers should take the number of filters as an argument, use a kernel size of 3 x 3, 'same' padding, and relu activation functions. The number of filters will double with each convolutional layer. The max pooling layers should have a pool size of 2 x 2. The intermediate dense layers before the final dense layer should take the number of nodes as an argument, use relu activation functions, and be followed by batch normalization. The final dense layer should have 10 nodes (= the number of classes in this laboration) and 'softmax' activation. Here we start with the Adam optimizer.

Relevant functions are

`model.add()`, adds a layer to the network

`Dense()`, a dense network layer

`Conv2D()`, performs 2D convolutions with a number of filters with a certain size (e.g. 3 x 3).

`BatchNormalization()`, perform batch normalization

`MaxPooling2D()`, saves the max for a given pool size, results in down sampling

`Flatten()`, flatten a multi-channel tensor into a long vector

`model.compile()`, compile the model, add " metrics=['accuracy'] " to print the classification accuracy during the training

See https://keras.io/api/layers/core_layers/dense/ and https://keras.io/api/layers/reshaping_layers/flatten/ for information on how the `Dense()` and `Flatten()` functions work

See https://keras.io/layers/convolutional/ for information on how `Conv2D()` works

See https://keras.io/layers/pooling/ for information on how `MaxPooling2D()` works

Import a relevant cost function for multi-class classification from keras.losses (https://keras.io/losses/) , it relates to how many classes you have.

See the following links for how to compile, train and evaluate the model

https://keras.io/api/models/model_training_apis/#compile-method

https://keras.io/api/models/model_training_apis/#fit-method

https://keras.io/api/models/model_training_apis/#evaluate-method

In [ ]:
```
from keras.models import Sequential, Model
from keras.layers import Input, Conv2D, BatchNormalization, MaxPooling2D, Flatte
from tensorflow.keras.optimizers import Adam
from keras.losses import CategoricalCrossentropy

# Set seed from random number generator, for better comparisons
from numpy.random import seed
seed(123)

def build_CNN(input_shape, n_conv_layers=2, n_filters=16, n_dense_layers=0, n_nc

    # Setup a sequential model
    model = Sequential()

    # Add first convolutional layer to the model, requires input shape
    # model.add(Input(shape=input_shape))
    model.add(Conv2D(filters= (1)*n_filters, kernel_size=(3,3), padding='same',a
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2,2)))

    # Add remaining convolutional layers to the model, the number of filters sho
    for i in range(n_conv_layers-1):
        model.add(Conv2D(filters= (2**(i+1))*n_filters, kernel_size=(3,3), paddi
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2,2)))

    # Add flatten layer
    model.add(Flatten())

    # Add intermediate dense layers
    for i in range(n_dense_layers):
        model.add(Dense(n_nodes, activation='relu'))
        model.add(BatchNormalization())
```

```
        if use_dropout:
            if use_dropout == True:
                use_dropout = 0.5
            model.add(Dropout(rate = use_dropout))

    # Add final dense layer
    model.add(Dense(10,activation='softmax'))

    # Compile model
    optimizer = Adam(learning_rate=learning_rate)

    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=


    return model
```

In [ ]:
```
# Lets define a help function for plotting the training results
import matplotlib.pyplot as plt
def plot_results(history):

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

    plt.figure(figsize=(10,4))
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.plot(loss)
    plt.plot(val_loss)
    plt.legend(['Training','Validation'])

    plt.figure(figsize=(10,4))
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.plot(acc)
    plt.plot(val_acc)
    plt.legend(['Training','Validation'])

    plt.show()
```

Part 11: Train 2D CNN

Time to train the 2D CNN, start with 2 convolutional layers, no intermediate dense layers, learning rate = 0.01. The first convolutional layer should have 16 filters (which means that the second convolutional layer will have 32 filters).

Relevant functions

`build_CNN`, the function we defined in Part 10, call it with the parameters you want to use

`model.fit()`, train the model with some training data

`model.evaluate()`, apply the trained model to some test data

See the following links for how to train and evaluate the model

https://keras.io/api/models/model_training_apis/#fit-method

https://keras.io/api/models/model_training_apis/#evaluate-method

2 convolutional layers, no intermediate dense layers

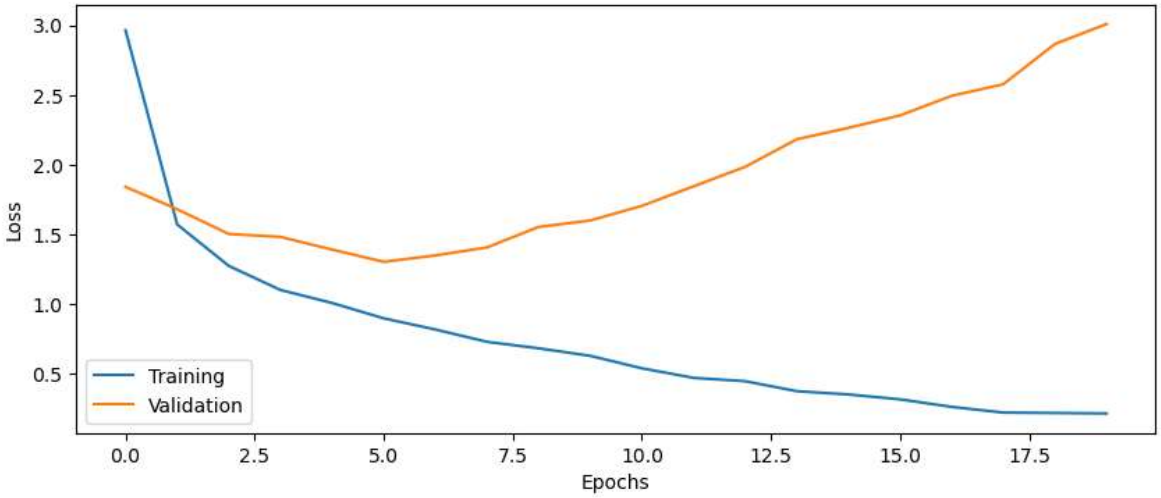
In [ ]:
```
# Setup some training parameters
batch_size = 100
epochs = 20
input_shape = Xtrain.shape[1:]

# Build model
model1 = build_CNN(input_shape,n_conv_layers=2,n_dense_layers=0)

# Train the model  using training data and validation data
history1 = model1.fit(Xtrain,Ytrain,epochs=epochs, batch_size=batch_size,validat
```

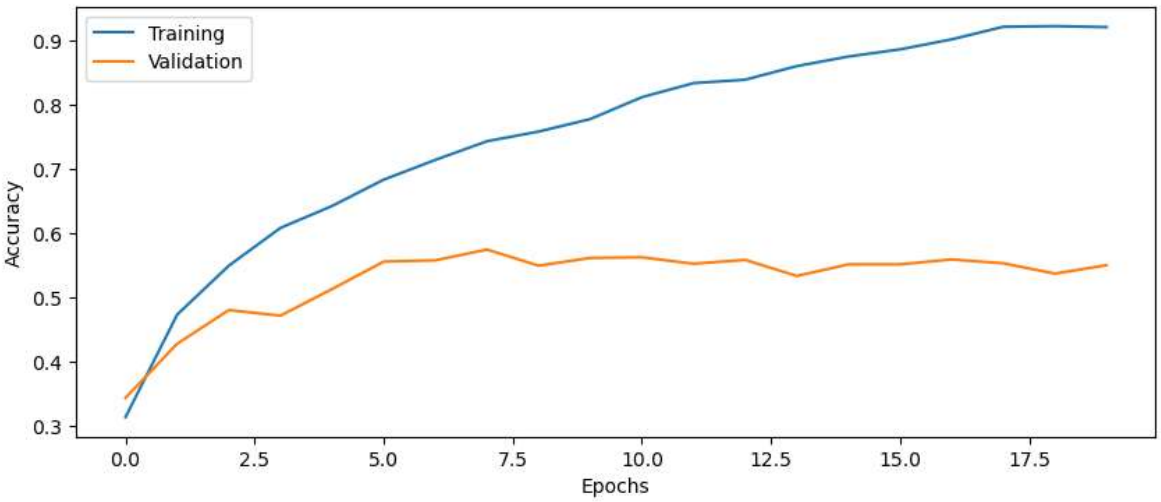
```
Epoch 1/20
70/70 [==============================] - 2s 32ms/step - loss: 2.9630 - accuracy:
0.3144 - val_loss: 1.8414 - val_accuracy: 0.3443
Epoch 2/20
70/70 [==============================] - 2s 30ms/step - loss: 1.5719 - accuracy:
0.4736 - val_loss: 1.6809 - val_accuracy: 0.4283
Epoch 3/20
70/70 [==============================] - 2s 29ms/step - loss: 1.2745 - accuracy:
0.5496 - val_loss: 1.5036 - val_accuracy: 0.4807
Epoch 4/20
70/70 [==============================] - 2s 30ms/step - loss: 1.1020 - accuracy:
0.6083 - val_loss: 1.4825 - val_accuracy: 0.4720
Epoch 5/20
70/70 [==============================] - 2s 29ms/step - loss: 1.0086 - accuracy:
0.6424 - val_loss: 1.3916 - val_accuracy: 0.5133
Epoch 6/20
70/70 [==============================] - 2s 29ms/step - loss: 0.8988 - accuracy:
0.6833 - val_loss: 1.3037 - val_accuracy: 0.5560
Epoch 7/20
70/70 [==============================] - 2s 29ms/step - loss: 0.8185 - accuracy:
0.7140 - val_loss: 1.3497 - val_accuracy: 0.5580
Epoch 8/20
70/70 [==============================] - 2s 29ms/step - loss: 0.7301 - accuracy:
0.7430 - val_loss: 1.4067 - val_accuracy: 0.5747
Epoch 9/20
70/70 [==============================] - 2s 29ms/step - loss: 0.6837 - accuracy:
0.7580 - val_loss: 1.5540 - val_accuracy: 0.5497
Epoch 10/20
70/70 [==============================] - 2s 30ms/step - loss: 0.6297 - accuracy:
0.7774 - val_loss: 1.6002 - val_accuracy: 0.5617
Epoch 11/20
70/70 [==============================] - 2s 30ms/step - loss: 0.5403 - accuracy:
0.8114 - val_loss: 1.7050 - val_accuracy: 0.5627
Epoch 12/20
70/70 [==============================] - 2s 31ms/step - loss: 0.4718 - accuracy:
0.8333 - val_loss: 1.8448 - val_accuracy: 0.5527
Epoch 13/20
70/70 [==============================] - 2s 32ms/step - loss: 0.4480 - accuracy:
0.8386 - val_loss: 1.9848 - val_accuracy: 0.5587
Epoch 14/20
70/70 [==============================] - 3s 44ms/step - loss: 0.3762 - accuracy:
0.8597 - val_loss: 2.1826 - val_accuracy: 0.5337
Epoch 15/20
70/70 [==============================] - 3s 43ms/step - loss: 0.3529 - accuracy:
0.8746 - val_loss: 2.2642 - val_accuracy: 0.5517
Epoch 16/20
70/70 [==============================] - 3s 46ms/step - loss: 0.3177 - accuracy:
0.8857 - val_loss: 2.3536 - val_accuracy: 0.5517
Epoch 17/20
70/70 [==============================] - 3s 46ms/step - loss: 0.2636 - accuracy:
0.9013 - val_loss: 2.4941 - val_accuracy: 0.5593
Epoch 18/20
70/70 [==============================] - 3s 46ms/step - loss: 0.2229 - accuracy:
0.9209 - val_loss: 2.5772 - val_accuracy: 0.5533
Epoch 19/20
70/70 [==============================] - 3s 43ms/step - loss: 0.2199 - accuracy:
0.9219 - val_loss: 2.8659 - val_accuracy: 0.5373
Epoch 20/20
70/70 [==============================] - 3s 43ms/step - loss: 0.2167 - accuracy:
0.9203 - val_loss: 3.0092 - val_accuracy: 0.5503
```

In [ ]:
```
# Evaluate the trained model on test set, not used in training or validation
score = model1.evaluate(Xtest,Ytest, batch_size=batch_size)
print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
20/20 [==============================] - 0s 9ms/step - loss: 3.0714 - accuracy:
0.5410
Test loss: 3.0714
Test accuracy: 0.5410
```

In [ ]:
```
# Plot the history from the training run
plot_results(history1)
```





In [ ]:
```
model1.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_24 (Conv2D)           (None, 32, 32, 16)        448
_________________________________________________________________
batch_normalization_24 (Batc (None, 32, 32, 16)        64
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 16, 16, 32)        4640
_________________________________________________________________
batch_normalization_25 (Batc (None, 16, 16, 32)        128
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 8, 8, 32)          0
_________________________________________________________________
flatten_10 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_12 (Dense)             (None, 10)                20490
=================================================================
Total params: 25,770
Trainable params: 25,674
Non-trainable params: 96
_________________________________________________________________
```

Part 12: Improving performance

Write down the test accuracy, are you satisfied with the classifier performance (random chance is 10%) ?

- Test accuracy: 0.5410
- better than random chance, but still far away from satisfing. severe overfitting needs to be mitigated

Question 10: How big is the difference between training and test accuracy?

- training accuracy: 0.9203
- difference: 0.3793

Question 11: For the DNN laboration we used a batch size of 10,000, why do we need to use a smaller batch size in this laboration?

- there're much more parameters than DNN, which needs more space occupied in the momory under same batch size.

2 convolutional layers, 1 intermediate dense layer (50 nodes)

In [ ]:
```
# Setup some training parameters
batch_size = 100
epochs = 20
input_shape = Xtrain.shape[1:]

# Build model
```

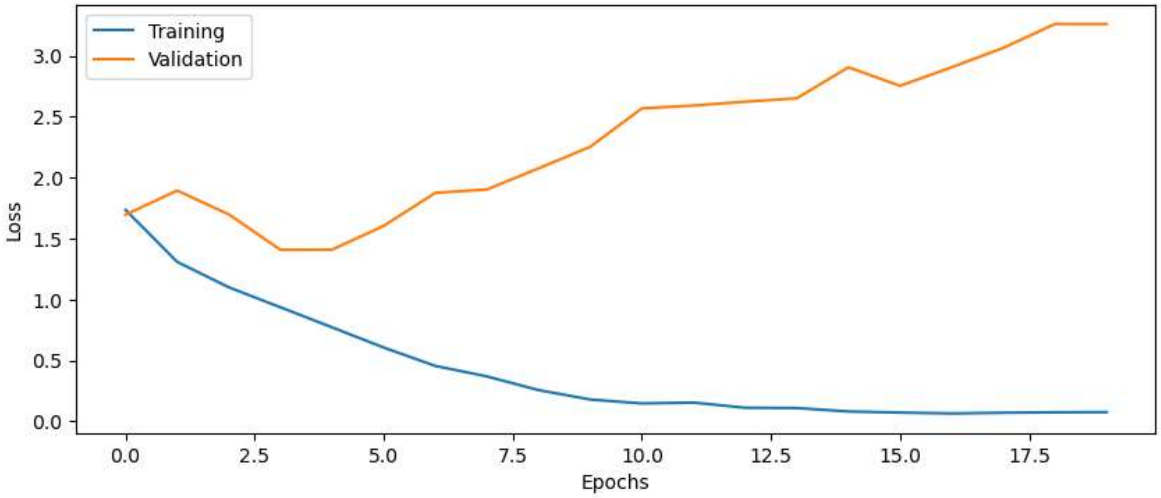
```
model2 = build_CNN(input_shape,n_conv_layers=2,n_dense_layers=1,n_nodes=50)

# Train the model  using training data and validation data
history2 = model2.fit(Xtrain,Ytrain,epochs=epochs, batch_size=batch_size,validat
```

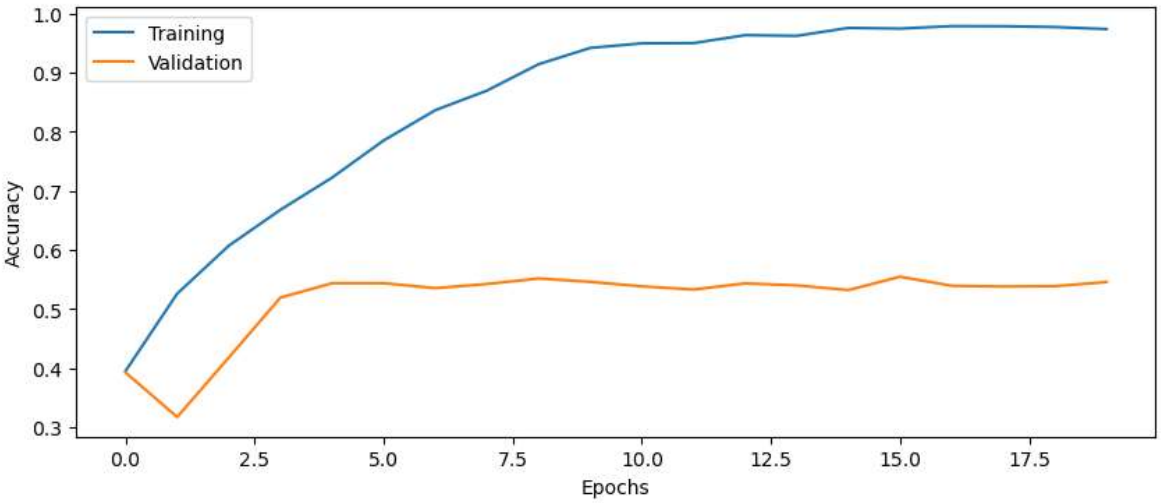
```
Epoch 1/20
70/70 [==============================] - 2s 33ms/step - loss: 1.7343 - accuracy:
0.3956 - val_loss: 1.6955 - val_accuracy: 0.3930
Epoch 2/20
70/70 [==============================] - 2s 30ms/step - loss: 1.3079 - accuracy:
0.5263 - val_loss: 1.8931 - val_accuracy: 0.3177
Epoch 3/20
70/70 [==============================] - 2s 34ms/step - loss: 1.0990 - accuracy:
0.6074 - val_loss: 1.6976 - val_accuracy: 0.4183
Epoch 4/20
70/70 [==============================] - 3s 44ms/step - loss: 0.9352 - accuracy:
0.6681 - val_loss: 1.4069 - val_accuracy: 0.5197
Epoch 5/20
70/70 [==============================] - 3s 45ms/step - loss: 0.7706 - accuracy:
0.7226 - val_loss: 1.4082 - val_accuracy: 0.5440
Epoch 6/20
70/70 [==============================] - 3s 45ms/step - loss: 0.6050 - accuracy:
0.7854 - val_loss: 1.6035 - val_accuracy: 0.5440
Epoch 7/20
70/70 [==============================] - 3s 44ms/step - loss: 0.4541 - accuracy:
0.8366 - val_loss: 1.8743 - val_accuracy: 0.5357
Epoch 8/20
70/70 [==============================] - 3s 44ms/step - loss: 0.3677 - accuracy:
0.8697 - val_loss: 1.9022 - val_accuracy: 0.5427
Epoch 9/20
70/70 [==============================] - 3s 46ms/step - loss: 0.2557 - accuracy:
0.9143 - val_loss: 2.0755 - val_accuracy: 0.5520
Epoch 10/20
70/70 [==============================] - 3s 45ms/step - loss: 0.1784 - accuracy:
0.9419 - val_loss: 2.2522 - val_accuracy: 0.5463
Epoch 11/20
70/70 [==============================] - 3s 44ms/step - loss: 0.1463 - accuracy:
0.9496 - val_loss: 2.5677 - val_accuracy: 0.5387
Epoch 12/20
70/70 [==============================] - 3s 42ms/step - loss: 0.1534 - accuracy:
0.9500 - val_loss: 2.5906 - val_accuracy: 0.5333
Epoch 13/20
70/70 [==============================] - 3s 44ms/step - loss: 0.1105 - accuracy:
0.9634 - val_loss: 2.6224 - val_accuracy: 0.5437
Epoch 14/20
70/70 [==============================] - 3s 42ms/step - loss: 0.1082 - accuracy:
0.9623 - val_loss: 2.6502 - val_accuracy: 0.5403
Epoch 15/20
70/70 [==============================] - 3s 42ms/step - loss: 0.0803 - accuracy:
0.9757 - val_loss: 2.9046 - val_accuracy: 0.5323
Epoch 16/20
70/70 [==============================] - 3s 42ms/step - loss: 0.0711 - accuracy:
0.9746 - val_loss: 2.7528 - val_accuracy: 0.5550
Epoch 17/20
70/70 [==============================] - 3s 42ms/step - loss: 0.0642 - accuracy:
0.9787 - val_loss: 2.9048 - val_accuracy: 0.5397
Epoch 18/20
70/70 [==============================] - 3s 42ms/step - loss: 0.0699 - accuracy:
0.9786 - val_loss: 3.0636 - val_accuracy: 0.5383
Epoch 19/20
70/70 [==============================] - 3s 42ms/step - loss: 0.0734 - accuracy:
0.9771 - val_loss: 3.2601 - val_accuracy: 0.5390
Epoch 20/20
70/70 [==============================] - 3s 42ms/step - loss: 0.0748 - accuracy:
0.9737 - val_loss: 3.2597 - val_accuracy: 0.5460
```

In [ ]:
```
# Evaluate the trained model on test set, not used in training or validation
score = model2.evaluate(Xtest,Ytest, batch_size=batch_size)
print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
20/20 [==============================] - 0s 6ms/step - loss: 3.2746 - accuracy:
0.5555
Test loss: 3.2746
Test accuracy: 0.5555
```

In [ ]:
```
# Plot the history from the training run
plot_results(history2)
```





4 convolutional layers, 1 intermediate dense layer (50 nodes)

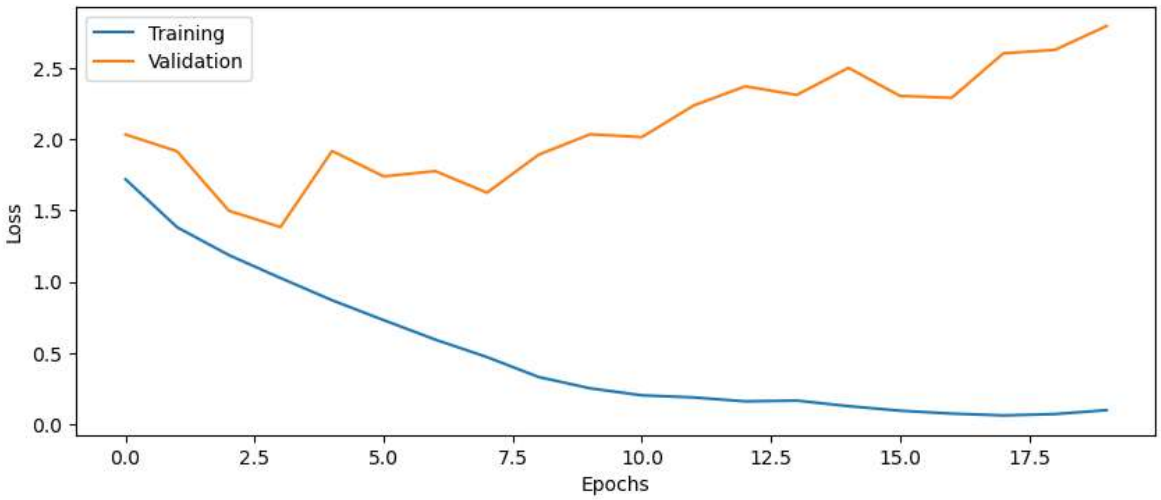
In [ ]:
```
# Setup some training parameters
batch_size = 100
epochs = 20
input_shape = Xtrain.shape[1:]

# Build model
model3 = build_CNN(input_shape,n_conv_layers=4,n_dense_layers=1,n_nodes=50)

# Train the model  using training data and validation data
history3 = model3.fit(Xtrain,Ytrain,epochs=epochs, batch_size=batch_size,validat
```

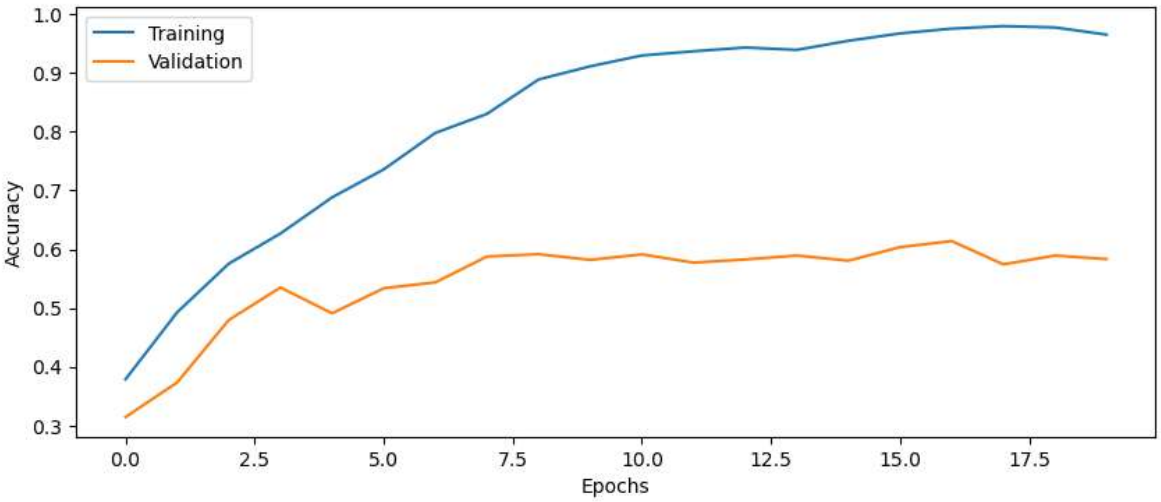
```
Epoch 1/20
70/70 [==============================] - 3s 45ms/step - loss: 1.7179 - accuracy:
0.3787 - val_loss: 2.0322 - val_accuracy: 0.3143
Epoch 2/20
70/70 [==============================] - 3s 41ms/step - loss: 1.3812 - accuracy:
0.4927 - val_loss: 1.9142 - val_accuracy: 0.3733
Epoch 3/20
70/70 [==============================] - 5s 67ms/step - loss: 1.1865 - accuracy:
0.5756 - val_loss: 1.4977 - val_accuracy: 0.4797
Epoch 4/20
70/70 [==============================] - 4s 62ms/step - loss: 1.0256 - accuracy:
0.6269 - val_loss: 1.3828 - val_accuracy: 0.5350
Epoch 5/20
70/70 [==============================] - 4s 63ms/step - loss: 0.8694 - accuracy:
0.6884 - val_loss: 1.9168 - val_accuracy: 0.4910
Epoch 6/20
70/70 [==============================] - 4s 63ms/step - loss: 0.7299 - accuracy:
0.7360 - val_loss: 1.7385 - val_accuracy: 0.5337
Epoch 7/20
70/70 [==============================] - 5s 71ms/step - loss: 0.5932 - accuracy:
0.7981 - val_loss: 1.7753 - val_accuracy: 0.5437
Epoch 8/20
70/70 [==============================] - 4s 64ms/step - loss: 0.4708 - accuracy:
0.8306 - val_loss: 1.6241 - val_accuracy: 0.5877
Epoch 9/20
70/70 [==============================] - 4s 61ms/step - loss: 0.3309 - accuracy:
0.8891 - val_loss: 1.8901 - val_accuracy: 0.5917
Epoch 10/20
70/70 [==============================] - 5s 70ms/step - loss: 0.2513 - accuracy:
0.9116 - val_loss: 2.0340 - val_accuracy: 0.5820
Epoch 11/20
70/70 [==============================] - 5s 65ms/step - loss: 0.2025 - accuracy:
0.9300 - val_loss: 2.0139 - val_accuracy: 0.5913
Epoch 12/20
70/70 [==============================] - 5s 71ms/step - loss: 0.1876 - accuracy:
0.9371 - val_loss: 2.2349 - val_accuracy: 0.5773
Epoch 13/20
70/70 [==============================] - 5s 67ms/step - loss: 0.1603 - accuracy:
0.9436 - val_loss: 2.3703 - val_accuracy: 0.5827
Epoch 14/20
70/70 [==============================] - 5s 67ms/step - loss: 0.1653 - accuracy:
0.9396 - val_loss: 2.3099 - val_accuracy: 0.5893
Epoch 15/20
70/70 [==============================] - 5s 67ms/step - loss: 0.1266 - accuracy:
0.9550 - val_loss: 2.4997 - val_accuracy: 0.5807
Epoch 16/20
70/70 [==============================] - 5s 74ms/step - loss: 0.0946 - accuracy:
0.9676 - val_loss: 2.3029 - val_accuracy: 0.6037
Epoch 17/20
70/70 [==============================] - 5s 72ms/step - loss: 0.0744 - accuracy:
0.9757 - val_loss: 2.2891 - val_accuracy: 0.6140
Epoch 18/20
70/70 [==============================] - 5s 74ms/step - loss: 0.0614 - accuracy:
0.9800 - val_loss: 2.6015 - val_accuracy: 0.5743
Epoch 19/20
70/70 [==============================] - 5s 68ms/step - loss: 0.0710 - accuracy:
0.9777 - val_loss: 2.6258 - val_accuracy: 0.5893
Epoch 20/20
70/70 [==============================] - 5s 74ms/step - loss: 0.0983 - accuracy:
0.9656 - val_loss: 2.7934 - val_accuracy: 0.5833
```

In [ ]:
```
# Evaluate the trained model on test set, not used in training or validation
score = model3.evaluate(Xtest,Ytest, batch_size=batch_size)
print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
20/20 [==============================] - 0s 14ms/step - loss: 2.7582 - accuracy:
0.5745
Test loss: 2.7582
Test accuracy: 0.5745
```

In [ ]:
```
# Plot the history from the training run
plot_results(history3)
```





Part 13: Plot the CNN architecture

To understand your network better, print the architecture using `model.summary()`

Question 12: How many trainable parameters does your network have? Which part of the network contains most of the parameters?

- Trainable params: 124,180
- the last conv2D layer contains the most of the parameters

Question 13: What is the input to and output of a Conv2D layer? What are the dimensions of the input and output?

- the input is the output of last (max_pooling2d) layer
- the output is the same image dimension as input but with a doubled channels for each pixel

Question 14: Is the batch size always the first dimension of each 4D tensor? Check the documentation for Conv2D, https://keras.io/layers/convolutional/

- yes, batch_shape is always the first dimension of each 4D tensor. Whether data_format='channels_last' or data_format='channels_first'

Question 15: If a convolutional layer that contains 128 filters is applied to an input with 32 channels, what is the number of channels in the output?

- 128

Question 16: Why is the number of parameters in each Conv2D layer *not* equal to the number of filters times the number of filter coefficients per filter (plus biases)?

- there are input channels to be considered too

Question 17: How does MaxPooling help in reducing the number of parameters to train?

- with a default stride(default to pool_size) and padding(default to "valid"), it will pooling each 2x2 pixels into 1x1 non-overlappingly. and make 2nx2n image size dimensions reduced to nxn

In [ ]:
```
# Print network architecture

model3.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_28 (Conv2D)           (None, 32, 32, 16)        448
_________________________________________________________________
batch_normalization_29 (Batc (None, 32, 32, 16)        64
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 16, 16, 32)        4640
_________________________________________________________________
batch_normalization_30 (Batc (None, 16, 16, 32)        128
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 8, 8, 64)          18496
_________________________________________________________________
batch_normalization_31 (Batc (None, 8, 8, 64)          256
_________________________________________________________________
max_pooling2d_28 (MaxPooling (None, 4, 4, 64)          0
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 4, 4, 128)         73856
_________________________________________________________________
batch_normalization_32 (Batc (None, 4, 4, 128)         512
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 2, 2, 128)         0
_________________________________________________________________
flatten_12 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_15 (Dense)             (None, 50)                25650
_________________________________________________________________
batch_normalization_33 (Batc (None, 50)                200
_________________________________________________________________
dense_16 (Dense)             (None, 10)                510
=================================================================
Total params: 124,760
Trainable params: 124,180
Non-trainable params: 580
_________________________________________________________________
```

Part 14: Dropout regularization

Add dropout regularization between each intermediate dense layer, dropout probability 50%.

Question 18: How much did the test accuracy improve with dropout, compared to without dropout?

- without : Test accuracy: 0.5745
- with : Test accuracy: 0.5780
- minor improve for this 2 runs, but can be better

Question 19: What other types of regularization can be applied? How can you add L2 regularization for the convolutional layers?

- L1/L2 regularization
- we can add augment kernel_regularizer/bias_regularizer/activity_regularizer = 'l2' to conv2D

4 convolutional layers, 1 intermediate dense layer (50 nodes), dropout

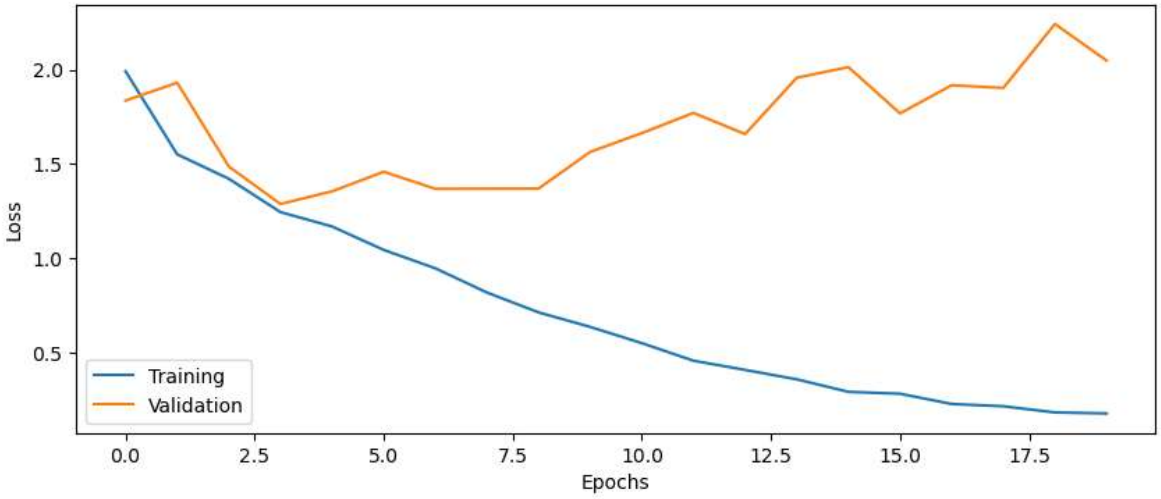
In [ ]:
```
# Setup some training parameters
batch_size = 100
epochs = 20
input_shape = Xtrain.shape[1:]

# Build model
model4 = build_CNN(input_shape,n_conv_layers=4,n_dense_layers=1,n_nodes=50, use_

# Train the model  using training data and validation data
history4 = model4.fit(Xtrain,Ytrain,epochs=epochs, batch_size=batch_size,validat
```

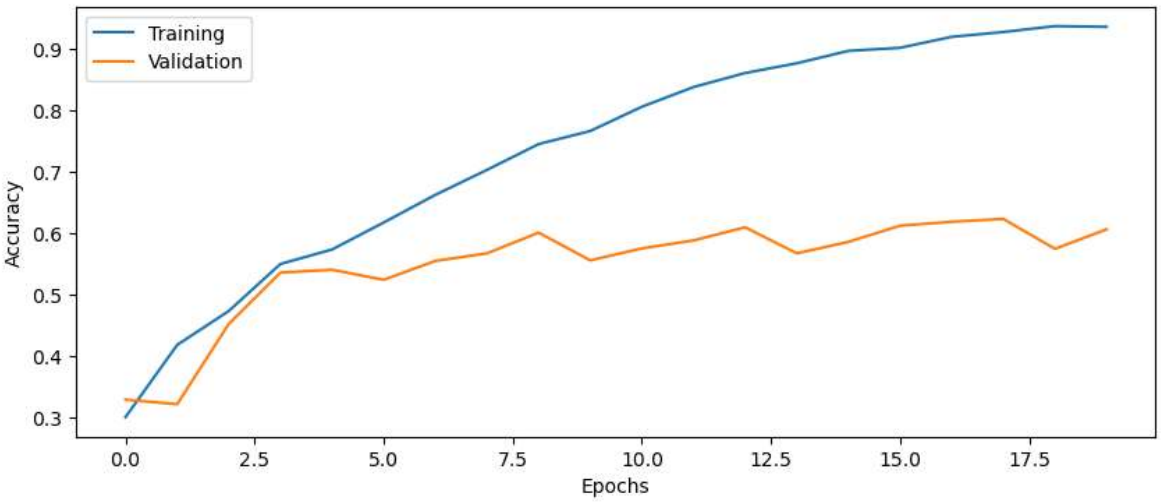
```
Epoch 1/20
70/70 [==============================] - 3s 43ms/step - loss: 1.9924 - accuracy:
0.3017 - val_loss: 1.8369 - val_accuracy: 0.3300
Epoch 2/20
70/70 [==============================] - 3s 43ms/step - loss: 1.5521 - accuracy:
0.4190 - val_loss: 1.9312 - val_accuracy: 0.3227
Epoch 3/20
70/70 [==============================] - 3s 43ms/step - loss: 1.4224 - accuracy:
0.4743 - val_loss: 1.4875 - val_accuracy: 0.4533
Epoch 4/20
70/70 [==============================] - 3s 41ms/step - loss: 1.2459 - accuracy:
0.5507 - val_loss: 1.2886 - val_accuracy: 0.5367
Epoch 5/20
70/70 [==============================] - 3s 40ms/step - loss: 1.1694 - accuracy:
0.5743 - val_loss: 1.3559 - val_accuracy: 0.5413
Epoch 6/20
70/70 [==============================] - 3s 42ms/step - loss: 1.0454 - accuracy:
0.6181 - val_loss: 1.4596 - val_accuracy: 0.5250
Epoch 7/20
70/70 [==============================] - 3s 40ms/step - loss: 0.9480 - accuracy:
0.6631 - val_loss: 1.3693 - val_accuracy: 0.5557
Epoch 8/20
70/70 [==============================] - 3s 41ms/step - loss: 0.8194 - accuracy:
0.7037 - val_loss: 1.3698 - val_accuracy: 0.5680
Epoch 9/20
70/70 [==============================] - 3s 42ms/step - loss: 0.7143 - accuracy:
0.7457 - val_loss: 1.3705 - val_accuracy: 0.6017
Epoch 10/20
70/70 [==============================] - 4s 57ms/step - loss: 0.6373 - accuracy:
0.7670 - val_loss: 1.5650 - val_accuracy: 0.5567
Epoch 11/20
70/70 [==============================] - 4s 63ms/step - loss: 0.5513 - accuracy:
0.8064 - val_loss: 1.6647 - val_accuracy: 0.5760
Epoch 12/20
70/70 [==============================] - 5s 65ms/step - loss: 0.4580 - accuracy:
0.8384 - val_loss: 1.7715 - val_accuracy: 0.5890
Epoch 13/20
70/70 [==============================] - 5s 65ms/step - loss: 0.4095 - accuracy:
0.8613 - val_loss: 1.6595 - val_accuracy: 0.6103
Epoch 14/20
70/70 [==============================] - 5s 67ms/step - loss: 0.3596 - accuracy:
0.8771 - val_loss: 1.9579 - val_accuracy: 0.5680
Epoch 15/20
70/70 [==============================] - 5s 67ms/step - loss: 0.2933 - accuracy:
0.8973 - val_loss: 2.0142 - val_accuracy: 0.5867
Epoch 16/20
70/70 [==============================] - 4s 61ms/step - loss: 0.2833 - accuracy:
0.9023 - val_loss: 1.7690 - val_accuracy: 0.6130
Epoch 17/20
70/70 [==============================] - 5s 65ms/step - loss: 0.2290 - accuracy:
0.9201 - val_loss: 1.9182 - val_accuracy: 0.6193
Epoch 18/20
70/70 [==============================] - 4s 61ms/step - loss: 0.2168 - accuracy:
0.9280 - val_loss: 1.9038 - val_accuracy: 0.6240
Epoch 19/20
70/70 [==============================] - 4s 63ms/step - loss: 0.1842 - accuracy:
0.9376 - val_loss: 2.2428 - val_accuracy: 0.5753
Epoch 20/20
70/70 [==============================] - 4s 61ms/step - loss: 0.1789 - accuracy:
0.9364 - val_loss: 2.0490 - val_accuracy: 0.6070
```

In [ ]:
```
# Evaluate the trained model on test set, not used in training or validation
score = model4.evaluate(Xtest,Ytest, batch_size=batch_size)
print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
200/200 [==============================] - 1s 3ms/step - loss: 2.2668 - accuracy:
0.5780
Test loss: 2.2668
Test accuracy: 0.5780
```

In [ ]:
```
# Plot the history from the training run
plot_results(history4)
```





Part 15: Tweaking performance

You have now seen the basic building blocks of a 2D CNN. To further improve performance involves changing the number of convolutional layers, the number of filters per layer, the number of intermediate dense layers, the number of nodes in the intermediate dense layers, batch size, learning rate, number of epochs, etc. Spend some time (30 - 90 minutes) testing different settings.

Question 20: How high test accuracy can you obtain? What is your best configuration?

- Test accuracy: 0.4895

- higher acuracy can be expected with more epochs(and maybe a lower learning rate) but no further experiment here due to time constraints.
- parameters:
- batch_size = 5
- epochs = 40
- n_conv_layers=4,n_dense_layers=2,n_nodes=200, use_dropout=0.6

Your best config

In [ ]:
```
# Setup some training parameters
batch_size = 5
epochs = 40
input_shape = Xtrain.shape[1:]

# Build model
model5 = build_CNN(input_shape,n_conv_layers=4,n_dense_layers=2,n_nodes=200, use

# Train the model  using training data and validation data
history5 = model5.fit(Xtrain,Ytrain,epochs=epochs, batch_size=batch_size,validat
```

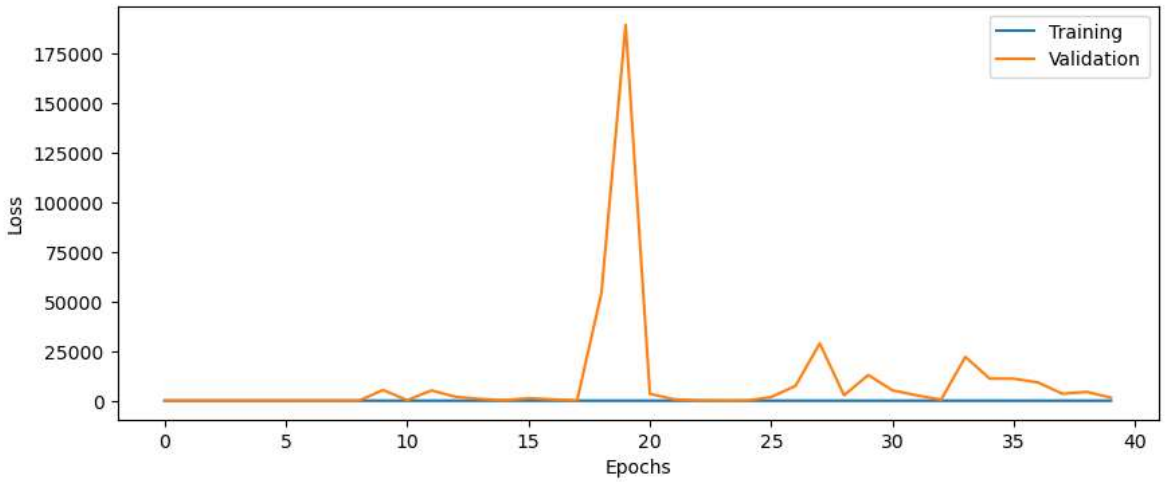
```
Epoch 1/40
1400/1400 [==============================] - 8s 5ms/step - loss: 2.4898 - accurac
y: 0.1644 - val_loss: 2.0614 - val_accuracy: 0.2337
Epoch 2/40
1400/1400 [==============================] - 7s 5ms/step - loss: 2.1917 - accurac
y: 0.1814 - val_loss: 2.0129 - val_accuracy: 0.2677
Epoch 3/40
1400/1400 [==============================] - 7s 5ms/step - loss: 2.1718 - accurac
y: 0.1971 - val_loss: 2.0047 - val_accuracy: 0.2250
Epoch 4/40
1400/1400 [==============================] - 7s 5ms/step - loss: 2.1614 - accurac
y: 0.2001 - val_loss: 2.1590 - val_accuracy: 0.2050
Epoch 5/40
1400/1400 [==============================] - 7s 5ms/step - loss: 2.1593 - accurac
y: 0.2056 - val_loss: 2.0499 - val_accuracy: 0.2447
Epoch 6/40
1400/1400 [==============================] - 7s 5ms/step - loss: 2.0884 - accurac
y: 0.2349 - val_loss: 1.9470 - val_accuracy: 0.2613
Epoch 7/40
1400/1400 [==============================] - 7s 5ms/step - loss: 2.1159 - accurac
y: 0.2204 - val_loss: 18.5209 - val_accuracy: 0.2993
Epoch 8/40
1400/1400 [==============================] - 7s 5ms/step - loss: 2.0717 - accurac
y: 0.2473 - val_loss: 1.9748 - val_accuracy: 0.2680
Epoch 9/40
1400/1400 [==============================] - 7s 5ms/step - loss: 2.0604 - accurac
y: 0.2393 - val_loss: 2.1006 - val_accuracy: 0.2970
Epoch 10/40
1400/1400 [==============================] - 7s 5ms/step - loss: 2.0615 - accurac
y: 0.2511 - val_loss: 5370.9146 - val_accuracy: 0.3203
Epoch 11/40
1400/1400 [==============================] - 7s 5ms/step - loss: 2.0214 - accurac
y: 0.2606 - val_loss: 252.0111 - val_accuracy: 0.2737
Epoch 12/40
1400/1400 [==============================] - 7s 5ms/step - loss: 2.0337 - accurac
y: 0.2519 - val_loss: 5160.2085 - val_accuracy: 0.3257
Epoch 13/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.9860 - accurac
y: 0.2753 - val_loss: 1938.1532 - val_accuracy: 0.3563
Epoch 14/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.9694 - accurac
y: 0.2801 - val_loss: 907.1464 - val_accuracy: 0.3057
Epoch 15/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.9894 - accurac
y: 0.2697 - val_loss: 198.2564 - val_accuracy: 0.3713
Epoch 16/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.9864 - accurac
y: 0.2816 - val_loss: 1238.2584 - val_accuracy: 0.3043
Epoch 17/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.9693 - accurac
y: 0.2736 - val_loss: 639.0750 - val_accuracy: 0.3577
Epoch 18/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.9409 - accurac
y: 0.2954 - val_loss: 96.1956 - val_accuracy: 0.4053
Epoch 19/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.9352 - accurac
y: 0.2891 - val_loss: 54106.6016 - val_accuracy: 0.3140
Epoch 20/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.9173 - accurac
y: 0.3007 - val_loss: 189331.2031 - val_accuracy: 0.4207
```

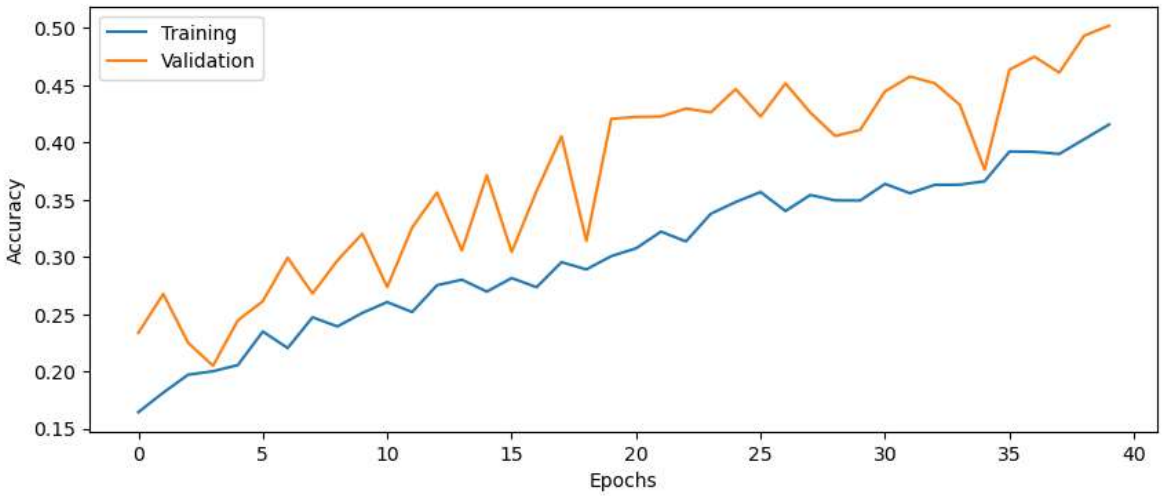
```
Epoch 21/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.8927 - accurac
y: 0.3076 - val_loss: 3444.4517 - val_accuracy: 0.4223
Epoch 22/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.8671 - accurac
y: 0.3221 - val_loss: 680.2239 - val_accuracy: 0.4227
Epoch 23/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.8773 - accurac
y: 0.3136 - val_loss: 129.1190 - val_accuracy: 0.4297
Epoch 24/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.8435 - accurac
y: 0.3377 - val_loss: 6.9771 - val_accuracy: 0.4263
Epoch 25/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.8253 - accurac
y: 0.3480 - val_loss: 47.3917 - val_accuracy: 0.4467
Epoch 26/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.8096 - accurac
y: 0.3567 - val_loss: 1847.9668 - val_accuracy: 0.4227
Epoch 27/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.8333 - accurac
y: 0.3401 - val_loss: 7444.1162 - val_accuracy: 0.4517
Epoch 28/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.8103 - accurac
y: 0.3541 - val_loss: 28857.2266 - val_accuracy: 0.4260
Epoch 29/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.8082 - accurac
y: 0.3494 - val_loss: 2727.8044 - val_accuracy: 0.4057
Epoch 30/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.8050 - accurac
y: 0.3493 - val_loss: 12930.0732 - val_accuracy: 0.4110
Epoch 31/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.7665 - accurac
y: 0.3639 - val_loss: 5210.2139 - val_accuracy: 0.4447
Epoch 32/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.8005 - accurac
y: 0.3557 - val_loss: 2674.0823 - val_accuracy: 0.4577
Epoch 33/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.7762 - accurac
y: 0.3630 - val_loss: 490.9593 - val_accuracy: 0.4517
Epoch 34/40
1400/1400 [==============================] - 8s 6ms/step - loss: 1.7818 - accurac
y: 0.3631 - val_loss: 22097.7129 - val_accuracy: 0.4330
Epoch 35/40
1400/1400 [==============================] - 8s 6ms/step - loss: 1.7683 - accurac
y: 0.3661 - val_loss: 11146.5283 - val_accuracy: 0.3763
Epoch 36/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.7373 - accurac
y: 0.3921 - val_loss: 11078.5244 - val_accuracy: 0.4637
Epoch 37/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.7181 - accurac
y: 0.3919 - val_loss: 9155.6963 - val_accuracy: 0.4750
Epoch 38/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.7091 - accurac
y: 0.3900 - val_loss: 3478.7480 - val_accuracy: 0.4610
Epoch 39/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.6908 - accurac
y: 0.4029 - val_loss: 4432.7393 - val_accuracy: 0.4933
Epoch 40/40
1400/1400 [==============================] - 7s 5ms/step - loss: 1.6585 - accurac
y: 0.4157 - val_loss: 1513.3977 - val_accuracy: 0.5020
```

In [ ]:
```
# Evaluate the trained model on test set, not used in training or validation
score = model5.evaluate(Xtest,Ytest, batch_size=batch_size)
print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
400/400 [==============================] - 0s 1ms/step - loss: 1006.8397 - accura
cy: 0.4895
Test loss: 1006.8397
Test accuracy: 0.4895
```

In [ ]:
```
# Plot the history from the training run
plot_results(history5)
```





Part 16: Rotate the test images

How high is the test accuracy if we rotate the test images? In other words, how good is the CNN at generalizing to rotated images?

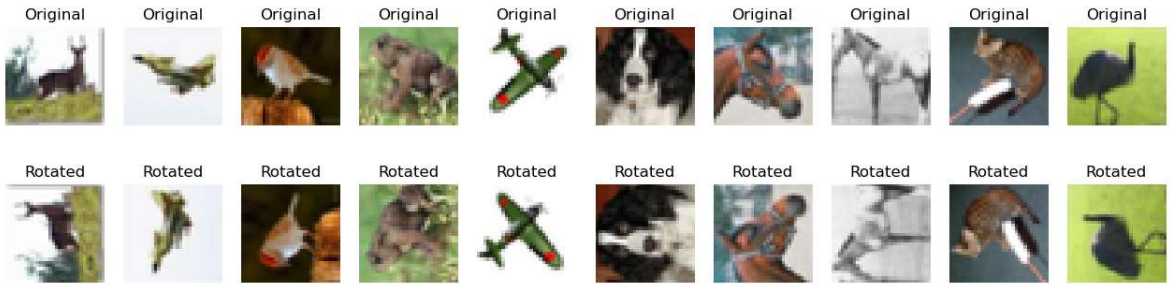
Rotate each test image 90 degrees, the cells are already finished.

Question 21: What is the test accuracy for rotated test images, compared to test images without rotation? Explain the difference in accuracy.

In [ ]:
```
def myrotate(images):

    images_rot = np.rot90(images, axes=(1,2))
```

```
    return images_rot
```

In [ ]:
```
# Rotate the test images 90 degrees
Xtest_rotated = myrotate(Xtest)

# Look at some rotated images
plt.figure(figsize=(16,4))
for i in range(10):
    idx = np.random.randint(500)

    plt.subplot(2,10,i+1)
    plt.imshow(Xtest[idx]/2+0.5)
    plt.title("Original")
    plt.axis('off')

    plt.subplot(2,10,i+11)
    plt.imshow(Xtest_rotated[idx]/2+0.5)
    plt.title("Rotated")
    plt.axis('off')
plt.show()
```



In [ ]:
```
# Evaluate the trained model on rotated test set
score = model5.evaluate(Xtest_rotated,Ytest, batch_size=batch_size)
print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
400/400 [==============================] - 1s 1ms/step - loss: 1390.1122 - accura
cy: 0.2040
Test loss: 1390.1122
Test accuracy: 0.2040
```

Part 17: Augmentation using Keras `ImageDataGenerator`

We can increase the number of training images through data augmentation (we now ignore that CIFAR10 actually has 60 000 training images). Image augmentation is about creating similar images, by performing operations such as rotation, scaling, elastic deformations and flipping of existing images. This will prevent overfitting, especially if all the training images are in a certain orientation.

We will perform the augmentation on the fly, using a built-in function in Keras, called `ImageDataGenerator`

See
https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGene

, the .flow(x,y) functionality

Make sure to use different subsets for training and validation when you setup the flows, otherwise you will validate on the same data...

In [ ]:
```
# Get all 60 000 training images again. ImageDataGenerator manages validation da
(Xtrain, Ytrain), _ = cifar10.load_data()

# Reduce number of images to 10,000
Xtrain = Xtrain[0:10000]
Ytrain = Ytrain[0:10000]

# Change data type and rescale range
Xtrain = Xtrain.astype('float32')
Xtrain = Xtrain / 127.5 - 1

# Convert labels to hot encoding
Ytrain = to_categorical(Ytrain, 10)
```

In [ ]:
```
# Set up a data generator with on-the-fly data augmentation, 20% validation spli
# Use a rotation range of 30 degrees, horizontal and vertical flipping
from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(rotation_range=0.3,horizontal_flip=True, vertical_f

datagen.fit(Xtrain)

# Setup a flow for training data, assume that we can fit all images into CPU mem
datagen.flow(x=Xtrain,y=Ytrain, subset='training')

# Setup a flow for validation data, assume that we can fit all images into CPU m
datagen.flow(x=Xtrain,y=Ytrain, subset='validation')
```

Out[ ]:
```
<tensorflow.python.keras.preprocessing.image.NumpyArrayIterator at 0x2230297b48
8>
```

Part 18: What about big data?

Question 22: How would you change the code for the image generator if you cannot fit all training images in CPU memory? What is the disadvantage of doing that change?

In [ ]:
```
# Plot some augmented images
plot_datagen = datagen.flow(Xtrain, Ytrain, batch_size=1)

plt.figure(figsize=(12,4))
for i in range(18):
    (im, label) = plot_datagen.next()
    im = (im[0] + 1) * 127.5
    im = im.astype('int')
    label = np.flatnonzero(label)[0]

    plt.subplot(3,6,i+1)
    plt.tight_layout()
    plt.imshow(im)
    plt.title("Class: {} ({})".format(label, classes[label]))
```

```
    plt.axis('off')
plt.show()
```

Part 19: Train the CNN with images from the generator

See https://keras.io/api/models/model_training_apis/#fit-method for how to use model.fit with a generator instead of a fix dataset (numpy arrays)

To make the comparison fair to training without augmentation

```
steps_per_epoch should be set to: len(Xtrain)*(1 -
validation_split)/batch_size

validation_steps should be set to:
len(Xtrain)*validation_split/batch_size
```

This is required since with a generator, the fit function will not know how many examples your original dataset has.

Question 23: How quickly is the training accuracy increasing compared to without augmentation? Explain why there is a difference compared to without augmentation. We are here talking about the number of training epochs required to reach a certain accuracy, and not the training time in seconds. What parameter is necessary to change to perform more training?

- the training accuracy increasing much more slower than without augmentation
- the reversed/rotated images(but with same label) put an extra obstacle towards learning, which makes the model takes loger time to learn
- more epochs will perform more traings

Question 24: What other types of image augmentation can be applied, compared to what we use here?

- scale/translation/quality/color augmentation

In [ ]:
```
# Setup some training parameters
batch_size = 100
epochs = 200
```

```
input_shape = Xtrain.shape[1:]

# Build model (your best config)
# again for time considerations we are using:
# 4 convolutional layers, 1 intermediate dense layer (50 nodes), dropout
# and disscusstion/comparision is of course based on that
model6 = build_CNN(input_shape, n_conv_layers=4,n_dense_layers=1,n_nodes=50, use

validation_split=0.2

# Train the model using on the fly augmentation
history6 = model6.fit(datagen.flow(x=Xtrain,y=Ytrain, subset='training',batch_si
                      validation_data = datagen.flow(x=Xtrain,y=Ytrain, subset='
                      steps_per_epoch = len(Xtrain)*(1 - validation_split)/batch
                      validation_steps = len(Xtrain)*validation_split/batch_size
                      epochs=epochs)
```

```
Epoch 1/200
80/80 [==============================] - 4s 50ms/step - loss: 2.1481 - accuracy:
0.2387 - val_loss: 2.2292 - val_accuracy: 0.2430
Epoch 2/200
80/80 [==============================] - 4s 45ms/step - loss: 1.7491 - accuracy:
0.3293 - val_loss: 2.1895 - val_accuracy: 0.2545
Epoch 3/200
80/80 [==============================] - 4s 45ms/step - loss: 1.6269 - accuracy:
0.3954 - val_loss: 1.6957 - val_accuracy: 0.3675
Epoch 4/200
80/80 [==============================] - 4s 45ms/step - loss: 1.5441 - accuracy:
0.4322 - val_loss: 1.6020 - val_accuracy: 0.4025
Epoch 5/200
80/80 [==============================] - 4s 46ms/step - loss: 1.4593 - accuracy:
0.4624 - val_loss: 1.3447 - val_accuracy: 0.4975
Epoch 6/200
80/80 [==============================] - 4s 46ms/step - loss: 1.3854 - accuracy:
0.4850 - val_loss: 1.6763 - val_accuracy: 0.4440
Epoch 7/200
80/80 [==============================] - 4s 47ms/step - loss: 1.3463 - accuracy:
0.5013 - val_loss: 1.4715 - val_accuracy: 0.4800
Epoch 8/200
80/80 [==============================] - 4s 47ms/step - loss: 1.2945 - accuracy:
0.5259 - val_loss: 1.4521 - val_accuracy: 0.4865
Epoch 9/200
80/80 [==============================] - 4s 47ms/step - loss: 1.2459 - accuracy:
0.5409 - val_loss: 1.3712 - val_accuracy: 0.5300
Epoch 10/200
80/80 [==============================] - 4s 47ms/step - loss: 1.1914 - accuracy:
0.5740 - val_loss: 1.5243 - val_accuracy: 0.5160
Epoch 11/200
80/80 [==============================] - 4s 47ms/step - loss: 1.1635 - accuracy:
0.5756 - val_loss: 1.2681 - val_accuracy: 0.5745
Epoch 12/200
80/80 [==============================] - 4s 47ms/step - loss: 1.1284 - accuracy:
0.5991 - val_loss: 1.2532 - val_accuracy: 0.5615
Epoch 13/200
80/80 [==============================] - 4s 47ms/step - loss: 1.0820 - accuracy:
0.6173 - val_loss: 1.2729 - val_accuracy: 0.5605
Epoch 14/200
80/80 [==============================] - 4s 48ms/step - loss: 1.0674 - accuracy:
0.6189 - val_loss: 1.1886 - val_accuracy: 0.6055
Epoch 15/200
80/80 [==============================] - 4s 45ms/step - loss: 1.0101 - accuracy:
0.6424 - val_loss: 1.1851 - val_accuracy: 0.5630
Epoch 16/200
80/80 [==============================] - 4s 45ms/step - loss: 0.9984 - accuracy:
0.6474 - val_loss: 1.1611 - val_accuracy: 0.5945
Epoch 17/200
80/80 [==============================] - 4s 45ms/step - loss: 0.9628 - accuracy:
0.6579 - val_loss: 1.1838 - val_accuracy: 0.5795
Epoch 18/200
80/80 [==============================] - 4s 45ms/step - loss: 0.9529 - accuracy:
0.6603 - val_loss: 1.2688 - val_accuracy: 0.5800
Epoch 19/200
80/80 [==============================] - 4s 45ms/step - loss: 0.9076 - accuracy:
0.6755 - val_loss: 1.3559 - val_accuracy: 0.5755
Epoch 20/200
80/80 [==============================] - 4s 45ms/step - loss: 0.8784 - accuracy:
0.6880 - val_loss: 1.1618 - val_accuracy: 0.6070
```

```
Epoch 21/200
80/80 [==============================] - 4s 45ms/step - loss: 0.8597 - accuracy:
0.6961 - val_loss: 1.1749 - val_accuracy: 0.5890
Epoch 22/200
80/80 [==============================] - 4s 45ms/step - loss: 0.8626 - accuracy:
0.6967 - val_loss: 1.4687 - val_accuracy: 0.5575
Epoch 23/200
80/80 [==============================] - 4s 45ms/step - loss: 0.8242 - accuracy:
0.7090 - val_loss: 1.2530 - val_accuracy: 0.5945
Epoch 24/200
80/80 [==============================] - 4s 45ms/step - loss: 0.8107 - accuracy:
0.7186 - val_loss: 1.3061 - val_accuracy: 0.6070
Epoch 25/200
80/80 [==============================] - 4s 45ms/step - loss: 0.7735 - accuracy:
0.7264 - val_loss: 1.4420 - val_accuracy: 0.5810
Epoch 26/200
80/80 [==============================] - 4s 45ms/step - loss: 0.7723 - accuracy:
0.7261 - val_loss: 1.2270 - val_accuracy: 0.6095
Epoch 27/200
80/80 [==============================] - 4s 45ms/step - loss: 0.7431 - accuracy:
0.7437 - val_loss: 1.3428 - val_accuracy: 0.5895
Epoch 28/200
80/80 [==============================] - 4s 45ms/step - loss: 0.7421 - accuracy:
0.7391 - val_loss: 1.2493 - val_accuracy: 0.6055
Epoch 29/200
80/80 [==============================] - 4s 45ms/step - loss: 0.7121 - accuracy:
0.7514 - val_loss: 1.1765 - val_accuracy: 0.6190
Epoch 30/200
80/80 [==============================] - 4s 45ms/step - loss: 0.7140 - accuracy:
0.7520 - val_loss: 1.2952 - val_accuracy: 0.6165
Epoch 31/200
80/80 [==============================] - 4s 45ms/step - loss: 0.7002 - accuracy:
0.7526 - val_loss: 1.1602 - val_accuracy: 0.6360
Epoch 32/200
80/80 [==============================] - 4s 46ms/step - loss: 0.6622 - accuracy:
0.7654 - val_loss: 1.1375 - val_accuracy: 0.6410
Epoch 33/200
80/80 [==============================] - 4s 45ms/step - loss: 0.6443 - accuracy:
0.7711 - val_loss: 1.1374 - val_accuracy: 0.6500
Epoch 34/200
80/80 [==============================] - 4s 46ms/step - loss: 0.6142 - accuracy:
0.7855 - val_loss: 1.3269 - val_accuracy: 0.5985
Epoch 35/200
80/80 [==============================] - 4s 45ms/step - loss: 0.6401 - accuracy:
0.7779 - val_loss: 1.2567 - val_accuracy: 0.6220
Epoch 36/200
80/80 [==============================] - 4s 45ms/step - loss: 0.6110 - accuracy:
0.7904 - val_loss: 1.2168 - val_accuracy: 0.6235
Epoch 37/200
80/80 [==============================] - 4s 45ms/step - loss: 0.5840 - accuracy:
0.7956 - val_loss: 1.2814 - val_accuracy: 0.6220
Epoch 38/200
80/80 [==============================] - 4s 46ms/step - loss: 0.5834 - accuracy:
0.7981 - val_loss: 1.2486 - val_accuracy: 0.6425
Epoch 39/200
80/80 [==============================] - 4s 45ms/step - loss: 0.5722 - accuracy:
0.7972 - val_loss: 1.3265 - val_accuracy: 0.6455
Epoch 40/200
80/80 [==============================] - 4s 45ms/step - loss: 0.5410 - accuracy:
0.8092 - val_loss: 1.5027 - val_accuracy: 0.5850
```

```
Epoch 41/200
80/80 [==============================] - 4s 46ms/step - loss: 0.5403 - accuracy:
0.8164 - val_loss: 1.2223 - val_accuracy: 0.6455
Epoch 42/200
80/80 [==============================] - 4s 47ms/step - loss: 0.5212 - accuracy:
0.8154 - val_loss: 1.4194 - val_accuracy: 0.6175
Epoch 43/200
80/80 [==============================] - 4s 48ms/step - loss: 0.5016 - accuracy:
0.8226 - val_loss: 1.2956 - val_accuracy: 0.6380
Epoch 44/200
80/80 [==============================] - 4s 47ms/step - loss: 0.5059 - accuracy:
0.8269 - val_loss: 1.3882 - val_accuracy: 0.6205
Epoch 45/200
80/80 [==============================] - 4s 50ms/step - loss: 0.5258 - accuracy:
0.8179 - val_loss: 1.3525 - val_accuracy: 0.6190
Epoch 46/200
80/80 [==============================] - 5s 66ms/step - loss: 0.4764 - accuracy:
0.8321 - val_loss: 1.4256 - val_accuracy: 0.6175
Epoch 47/200
80/80 [==============================] - 6s 75ms/step - loss: 0.4735 - accuracy:
0.8336 - val_loss: 1.3621 - val_accuracy: 0.6290
Epoch 48/200
80/80 [==============================] - 6s 78ms/step - loss: 0.4594 - accuracy:
0.8422 - val_loss: 1.3265 - val_accuracy: 0.6285
Epoch 49/200
80/80 [==============================] - 6s 77ms/step - loss: 0.4366 - accuracy:
0.8497 - val_loss: 1.2735 - val_accuracy: 0.6345
Epoch 50/200
80/80 [==============================] - 6s 79ms/step - loss: 0.4566 - accuracy:
0.8489 - val_loss: 1.3812 - val_accuracy: 0.6220
Epoch 51/200
80/80 [==============================] - 6s 76ms/step - loss: 0.4201 - accuracy:
0.8551 - val_loss: 1.5807 - val_accuracy: 0.6235
Epoch 52/200
80/80 [==============================] - 6s 76ms/step - loss: 0.4255 - accuracy:
0.8534 - val_loss: 1.3980 - val_accuracy: 0.6435
Epoch 53/200
80/80 [==============================] - 6s 77ms/step - loss: 0.4240 - accuracy:
0.8551 - val_loss: 1.3448 - val_accuracy: 0.6335
Epoch 54/200
80/80 [==============================] - 6s 76ms/step - loss: 0.3968 - accuracy:
0.8614 - val_loss: 1.4926 - val_accuracy: 0.6160
Epoch 55/200
80/80 [==============================] - 6s 75ms/step - loss: 0.3982 - accuracy:
0.8618 - val_loss: 1.5909 - val_accuracy: 0.6245
Epoch 56/200
80/80 [==============================] - 6s 78ms/step - loss: 0.3828 - accuracy:
0.8705 - val_loss: 1.5985 - val_accuracy: 0.6135
Epoch 57/200
80/80 [==============================] - 6s 79ms/step - loss: 0.3863 - accuracy:
0.8655 - val_loss: 1.5493 - val_accuracy: 0.6145
Epoch 58/200
80/80 [==============================] - 6s 77ms/step - loss: 0.3838 - accuracy:
0.8676 - val_loss: 1.4398 - val_accuracy: 0.6360
Epoch 59/200
80/80 [==============================] - 6s 78ms/step - loss: 0.3810 - accuracy:
0.8730 - val_loss: 1.5388 - val_accuracy: 0.6080
Epoch 60/200
80/80 [==============================] - 6s 80ms/step - loss: 0.3719 - accuracy:
0.8774 - val_loss: 1.5519 - val_accuracy: 0.6315
```

```
Epoch 61/200
80/80 [==============================] - 6s 79ms/step - loss: 0.3655 - accuracy:
0.8784 - val_loss: 1.6098 - val_accuracy: 0.6120
Epoch 62/200
80/80 [==============================] - 6s 78ms/step - loss: 0.3477 - accuracy:
0.8801 - val_loss: 1.5251 - val_accuracy: 0.6420
Epoch 63/200
80/80 [==============================] - 6s 75ms/step - loss: 0.3239 - accuracy:
0.8844 - val_loss: 1.6232 - val_accuracy: 0.6375
Epoch 64/200
80/80 [==============================] - 6s 76ms/step - loss: 0.3374 - accuracy:
0.8855 - val_loss: 1.6702 - val_accuracy: 0.6260
Epoch 65/200
80/80 [==============================] - 6s 76ms/step - loss: 0.3321 - accuracy:
0.8848 - val_loss: 1.5070 - val_accuracy: 0.6325
Epoch 66/200
80/80 [==============================] - 6s 74ms/step - loss: 0.3311 - accuracy:
0.8839 - val_loss: 1.7914 - val_accuracy: 0.6065
Epoch 67/200
80/80 [==============================] - 6s 74ms/step - loss: 0.3177 - accuracy:
0.8889 - val_loss: 1.5957 - val_accuracy: 0.6325
Epoch 68/200
80/80 [==============================] - 6s 76ms/step - loss: 0.2958 - accuracy:
0.9020 - val_loss: 1.6958 - val_accuracy: 0.6235
Epoch 69/200
80/80 [==============================] - 6s 75ms/step - loss: 0.3120 - accuracy:
0.8906 - val_loss: 1.4702 - val_accuracy: 0.6615
Epoch 70/200
80/80 [==============================] - 6s 75ms/step - loss: 0.2992 - accuracy:
0.8995 - val_loss: 1.6636 - val_accuracy: 0.6230
Epoch 71/200
80/80 [==============================] - 6s 74ms/step - loss: 0.3029 - accuracy:
0.8967 - val_loss: 1.5482 - val_accuracy: 0.6315
Epoch 72/200
80/80 [==============================] - 6s 75ms/step - loss: 0.2891 - accuracy:
0.9009 - val_loss: 1.7181 - val_accuracy: 0.6110
Epoch 73/200
80/80 [==============================] - 6s 75ms/step - loss: 0.2865 - accuracy:
0.9001 - val_loss: 1.6387 - val_accuracy: 0.6295
Epoch 74/200
80/80 [==============================] - 6s 80ms/step - loss: 0.2785 - accuracy:
0.9061 - val_loss: 1.6696 - val_accuracy: 0.6180
Epoch 75/200
80/80 [==============================] - 6s 76ms/step - loss: 0.2915 - accuracy:
0.8999 - val_loss: 1.6582 - val_accuracy: 0.6325
Epoch 76/200
80/80 [==============================] - 6s 76ms/step - loss: 0.2703 - accuracy:
0.9071 - val_loss: 1.7066 - val_accuracy: 0.6285
Epoch 77/200
80/80 [==============================] - 6s 75ms/step - loss: 0.2783 - accuracy:
0.9041 - val_loss: 1.6723 - val_accuracy: 0.6190
Epoch 78/200
80/80 [==============================] - 6s 74ms/step - loss: 0.2581 - accuracy:
0.9095 - val_loss: 1.7333 - val_accuracy: 0.6290
Epoch 79/200
80/80 [==============================] - 6s 76ms/step - loss: 0.2743 - accuracy:
0.9054 - val_loss: 1.7287 - val_accuracy: 0.6210
Epoch 80/200
80/80 [==============================] - 6s 76ms/step - loss: 0.2951 - accuracy:
0.9010 - val_loss: 1.6225 - val_accuracy: 0.6545
```

```
Epoch 81/200
80/80 [==============================] - 6s 75ms/step - loss: 0.2528 - accuracy:
0.9186 - val_loss: 1.8201 - val_accuracy: 0.6285
Epoch 82/200
80/80 [==============================] - 6s 77ms/step - loss: 0.2540 - accuracy:
0.9119 - val_loss: 1.6674 - val_accuracy: 0.6280
Epoch 83/200
80/80 [==============================] - 6s 78ms/step - loss: 0.2471 - accuracy:
0.9159 - val_loss: 1.7605 - val_accuracy: 0.6380
Epoch 84/200
80/80 [==============================] - 6s 75ms/step - loss: 0.2558 - accuracy:
0.9109 - val_loss: 1.8126 - val_accuracy: 0.6195
Epoch 85/200
80/80 [==============================] - 6s 75ms/step - loss: 0.2292 - accuracy:
0.9212 - val_loss: 1.7857 - val_accuracy: 0.6340
Epoch 86/200
80/80 [==============================] - 6s 75ms/step - loss: 0.2230 - accuracy:
0.9226 - val_loss: 1.8972 - val_accuracy: 0.6325
Epoch 87/200
80/80 [==============================] - 6s 74ms/step - loss: 0.2296 - accuracy:
0.9218 - val_loss: 1.8111 - val_accuracy: 0.6330
Epoch 88/200
80/80 [==============================] - 6s 78ms/step - loss: 0.2231 - accuracy:
0.9220 - val_loss: 1.7335 - val_accuracy: 0.6445
Epoch 89/200
80/80 [==============================] - 6s 76ms/step - loss: 0.2290 - accuracy:
0.9218 - val_loss: 1.7306 - val_accuracy: 0.6335
Epoch 90/200
80/80 [==============================] - 6s 75ms/step - loss: 0.2140 - accuracy:
0.9277 - val_loss: 1.6817 - val_accuracy: 0.6530
Epoch 91/200
80/80 [==============================] - 6s 74ms/step - loss: 0.2359 - accuracy:
0.9199 - val_loss: 1.8042 - val_accuracy: 0.6370
Epoch 92/200
80/80 [==============================] - 6s 74ms/step - loss: 0.2272 - accuracy:
0.9244 - val_loss: 1.7850 - val_accuracy: 0.6520
Epoch 93/200
80/80 [==============================] - 6s 75ms/step - loss: 0.2210 - accuracy:
0.9240 - val_loss: 1.8832 - val_accuracy: 0.6310
Epoch 94/200
80/80 [==============================] - 6s 75ms/step - loss: 0.2166 - accuracy:
0.9245 - val_loss: 1.8547 - val_accuracy: 0.6235
Epoch 95/200
80/80 [==============================] - 6s 77ms/step - loss: 0.1945 - accuracy:
0.9325 - val_loss: 1.9568 - val_accuracy: 0.6205
Epoch 96/200
80/80 [==============================] - 6s 75ms/step - loss: 0.2252 - accuracy:
0.9247 - val_loss: 1.8276 - val_accuracy: 0.6390
Epoch 97/200
80/80 [==============================] - 6s 76ms/step - loss: 0.2092 - accuracy:
0.9270 - val_loss: 1.9261 - val_accuracy: 0.6230
Epoch 98/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1962 - accuracy:
0.9360 - val_loss: 1.9669 - val_accuracy: 0.6410
Epoch 99/200
80/80 [==============================] - 6s 77ms/step - loss: 0.1948 - accuracy:
0.9366 - val_loss: 2.0893 - val_accuracy: 0.6190
Epoch 100/200
80/80 [==============================] - 6s 78ms/step - loss: 0.1973 - accuracy:
0.9331 - val_loss: 1.8855 - val_accuracy: 0.6375
```

```
Epoch 101/200
80/80 [==============================] - 6s 79ms/step - loss: 0.1995 - accuracy:
0.9329 - val_loss: 2.0990 - val_accuracy: 0.6085
Epoch 102/200
80/80 [==============================] - 6s 79ms/step - loss: 0.2133 - accuracy:
0.9291 - val_loss: 1.8864 - val_accuracy: 0.6410
Epoch 103/200
80/80 [==============================] - 6s 79ms/step - loss: 0.1967 - accuracy:
0.9306 - val_loss: 1.8584 - val_accuracy: 0.6370
Epoch 104/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1934 - accuracy:
0.9310 - val_loss: 1.9576 - val_accuracy: 0.6275
Epoch 105/200
80/80 [==============================] - 6s 74ms/step - loss: 0.1861 - accuracy:
0.9335 - val_loss: 1.9700 - val_accuracy: 0.6255
Epoch 106/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1963 - accuracy:
0.9334 - val_loss: 2.2215 - val_accuracy: 0.6210
Epoch 107/200
80/80 [==============================] - 6s 75ms/step - loss: 0.1905 - accuracy:
0.9352 - val_loss: 2.0361 - val_accuracy: 0.6195
Epoch 108/200
80/80 [==============================] - 6s 76ms/step - loss: 0.2069 - accuracy:
0.9266 - val_loss: 1.9686 - val_accuracy: 0.6315
Epoch 109/200
80/80 [==============================] - 6s 75ms/step - loss: 0.1729 - accuracy:
0.9421 - val_loss: 1.8656 - val_accuracy: 0.6315
Epoch 110/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1951 - accuracy:
0.9319 - val_loss: 2.0003 - val_accuracy: 0.6300
Epoch 111/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1673 - accuracy:
0.9444 - val_loss: 2.1511 - val_accuracy: 0.6095
Epoch 112/200
80/80 [==============================] - 6s 78ms/step - loss: 0.1875 - accuracy:
0.9365 - val_loss: 2.0386 - val_accuracy: 0.6370
Epoch 113/200
80/80 [==============================] - 6s 78ms/step - loss: 0.1779 - accuracy:
0.9386 - val_loss: 2.3414 - val_accuracy: 0.6155
Epoch 114/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1750 - accuracy:
0.9395 - val_loss: 2.0626 - val_accuracy: 0.6300
Epoch 115/200
80/80 [==============================] - 6s 79ms/step - loss: 0.1666 - accuracy:
0.9449 - val_loss: 2.0578 - val_accuracy: 0.6315
Epoch 116/200
80/80 [==============================] - 7s 82ms/step - loss: 0.1833 - accuracy:
0.9361 - val_loss: 1.9394 - val_accuracy: 0.6375
Epoch 117/200
80/80 [==============================] - 7s 82ms/step - loss: 0.1755 - accuracy:
0.9419 - val_loss: 2.0950 - val_accuracy: 0.6280
Epoch 118/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1617 - accuracy:
0.9494 - val_loss: 1.9754 - val_accuracy: 0.6380
Epoch 119/200
80/80 [==============================] - 6s 78ms/step - loss: 0.1607 - accuracy:
0.9464 - val_loss: 1.9488 - val_accuracy: 0.6325
Epoch 120/200
80/80 [==============================] - 6s 81ms/step - loss: 0.1854 - accuracy:
0.9351 - val_loss: 1.9060 - val_accuracy: 0.6420
```

```
Epoch 121/200
80/80 [==============================] - 6s 79ms/step - loss: 0.1605 - accuracy:
0.9456 - val_loss: 1.9713 - val_accuracy: 0.6560
Epoch 122/200
80/80 [==============================] - 6s 77ms/step - loss: 0.1364 - accuracy:
0.9514 - val_loss: 2.2965 - val_accuracy: 0.6245
Epoch 123/200
80/80 [==============================] - 6s 77ms/step - loss: 0.1589 - accuracy:
0.9459 - val_loss: 2.1209 - val_accuracy: 0.6340
Epoch 124/200
80/80 [==============================] - 6s 78ms/step - loss: 0.1694 - accuracy:
0.9436 - val_loss: 2.0538 - val_accuracy: 0.6435
Epoch 125/200
80/80 [==============================] - 6s 79ms/step - loss: 0.1744 - accuracy:
0.9400 - val_loss: 2.0359 - val_accuracy: 0.6515
Epoch 126/200
80/80 [==============================] - 6s 79ms/step - loss: 0.1665 - accuracy:
0.9415 - val_loss: 1.9679 - val_accuracy: 0.6425
Epoch 127/200
80/80 [==============================] - 6s 77ms/step - loss: 0.1505 - accuracy:
0.9479 - val_loss: 2.0108 - val_accuracy: 0.6370
Epoch 128/200
80/80 [==============================] - 6s 75ms/step - loss: 0.1765 - accuracy:
0.9414 - val_loss: 1.9122 - val_accuracy: 0.6360
Epoch 129/200
80/80 [==============================] - 6s 78ms/step - loss: 0.1397 - accuracy:
0.9551 - val_loss: 2.0124 - val_accuracy: 0.6445
Epoch 130/200
80/80 [==============================] - 6s 78ms/step - loss: 0.1360 - accuracy:
0.9541 - val_loss: 2.0752 - val_accuracy: 0.6410
Epoch 131/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1622 - accuracy:
0.9420 - val_loss: 2.1293 - val_accuracy: 0.6555
Epoch 132/200
80/80 [==============================] - 6s 77ms/step - loss: 0.1555 - accuracy:
0.9471 - val_loss: 2.0900 - val_accuracy: 0.6410
Epoch 133/200
80/80 [==============================] - 6s 77ms/step - loss: 0.1383 - accuracy:
0.9504 - val_loss: 2.2992 - val_accuracy: 0.6300
Epoch 134/200
80/80 [==============================] - 6s 77ms/step - loss: 0.1341 - accuracy:
0.9548 - val_loss: 2.1792 - val_accuracy: 0.6315
Epoch 135/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1492 - accuracy:
0.9519 - val_loss: 2.1986 - val_accuracy: 0.6340
Epoch 136/200
80/80 [==============================] - 6s 77ms/step - loss: 0.1521 - accuracy:
0.9481 - val_loss: 2.0163 - val_accuracy: 0.6405
Epoch 137/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1414 - accuracy:
0.9516 - val_loss: 2.1294 - val_accuracy: 0.6335
Epoch 138/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1585 - accuracy:
0.9488 - val_loss: 2.1267 - val_accuracy: 0.6410
Epoch 139/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1536 - accuracy:
0.9492 - val_loss: 2.1909 - val_accuracy: 0.6270
Epoch 140/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1515 - accuracy:
0.9473 - val_loss: 2.2421 - val_accuracy: 0.6190
```

```
Epoch 141/200
80/80 [==============================] - 6s 75ms/step - loss: 0.1375 - accuracy:
0.9517 - val_loss: 2.1313 - val_accuracy: 0.6325
Epoch 142/200
80/80 [==============================] - 6s 74ms/step - loss: 0.1553 - accuracy:
0.9470 - val_loss: 2.0664 - val_accuracy: 0.6345
Epoch 143/200
80/80 [==============================] - 6s 75ms/step - loss: 0.1378 - accuracy:
0.9528 - val_loss: 2.0698 - val_accuracy: 0.6375
Epoch 144/200
80/80 [==============================] - 6s 74ms/step - loss: 0.1297 - accuracy:
0.9561 - val_loss: 2.0989 - val_accuracy: 0.6405
Epoch 145/200
80/80 [==============================] - 6s 77ms/step - loss: 0.1459 - accuracy:
0.9511 - val_loss: 2.2792 - val_accuracy: 0.6280
Epoch 146/200
80/80 [==============================] - 6s 77ms/step - loss: 0.1545 - accuracy:
0.9494 - val_loss: 2.1209 - val_accuracy: 0.6360
Epoch 147/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1249 - accuracy:
0.9566 - val_loss: 2.2791 - val_accuracy: 0.6315
Epoch 148/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1313 - accuracy:
0.9548 - val_loss: 2.1758 - val_accuracy: 0.6320
Epoch 149/200
80/80 [==============================] - 6s 75ms/step - loss: 0.1410 - accuracy:
0.9506 - val_loss: 2.2487 - val_accuracy: 0.6375
Epoch 150/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1304 - accuracy:
0.9548 - val_loss: 2.2415 - val_accuracy: 0.6215
Epoch 151/200
80/80 [==============================] - 6s 75ms/step - loss: 0.1167 - accuracy:
0.9603 - val_loss: 2.2515 - val_accuracy: 0.6340
Epoch 152/200
80/80 [==============================] - 6s 75ms/step - loss: 0.1342 - accuracy:
0.9520 - val_loss: 2.2349 - val_accuracy: 0.6225
Epoch 153/200
80/80 [==============================] - 6s 75ms/step - loss: 0.1264 - accuracy:
0.9556 - val_loss: 2.1618 - val_accuracy: 0.6435
Epoch 154/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1247 - accuracy:
0.9560 - val_loss: 2.4175 - val_accuracy: 0.6175
Epoch 155/200
80/80 [==============================] - 6s 75ms/step - loss: 0.1214 - accuracy:
0.9594 - val_loss: 2.2763 - val_accuracy: 0.6360
Epoch 156/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1316 - accuracy:
0.9539 - val_loss: 2.3727 - val_accuracy: 0.6180
Epoch 157/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1155 - accuracy:
0.9607 - val_loss: 2.2220 - val_accuracy: 0.6380
Epoch 158/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1329 - accuracy:
0.9540 - val_loss: 2.2487 - val_accuracy: 0.6255
Epoch 159/200
80/80 [==============================] - 6s 75ms/step - loss: 0.1503 - accuracy:
0.9525 - val_loss: 2.2595 - val_accuracy: 0.6205
Epoch 160/200
80/80 [==============================] - 6s 78ms/step - loss: 0.1262 - accuracy:
0.9585 - val_loss: 2.1307 - val_accuracy: 0.6280
```

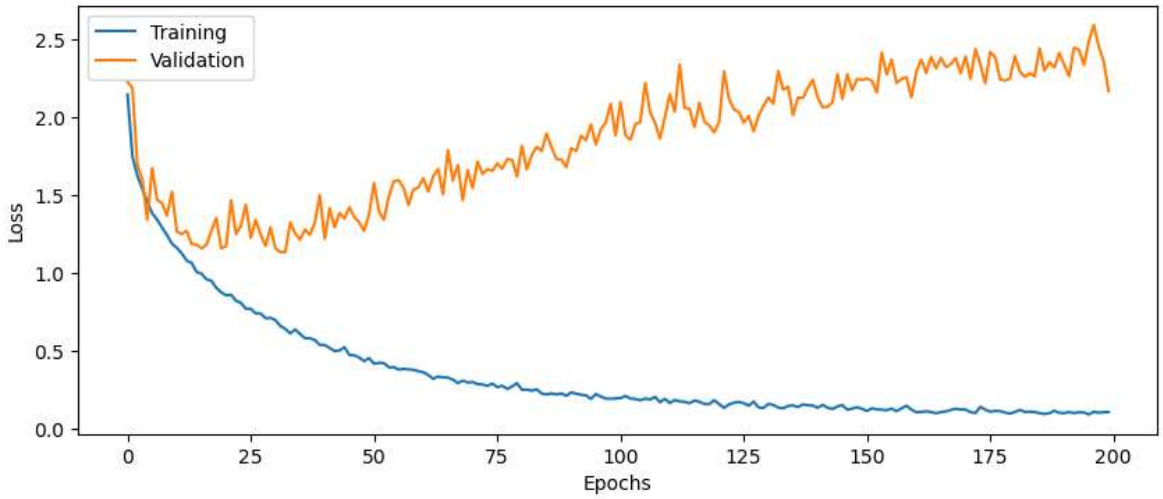
```
Epoch 161/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1083 - accuracy:
0.9629 - val_loss: 2.2934 - val_accuracy: 0.6385
Epoch 162/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1108 - accuracy:
0.9655 - val_loss: 2.3727 - val_accuracy: 0.6190
Epoch 163/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1150 - accuracy:
0.9622 - val_loss: 2.2875 - val_accuracy: 0.6450
Epoch 164/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1094 - accuracy:
0.9601 - val_loss: 2.3886 - val_accuracy: 0.6235
Epoch 165/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1023 - accuracy:
0.9659 - val_loss: 2.3170 - val_accuracy: 0.6400
Epoch 166/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1094 - accuracy:
0.9625 - val_loss: 2.3832 - val_accuracy: 0.6355
Epoch 167/200
80/80 [==============================] - 6s 74ms/step - loss: 0.1141 - accuracy:
0.9609 - val_loss: 2.3244 - val_accuracy: 0.6515
Epoch 168/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1233 - accuracy:
0.9588 - val_loss: 2.3432 - val_accuracy: 0.6260
Epoch 169/200
80/80 [==============================] - 6s 75ms/step - loss: 0.1304 - accuracy:
0.9559 - val_loss: 2.3819 - val_accuracy: 0.6330
Epoch 170/200
80/80 [==============================] - 6s 78ms/step - loss: 0.1258 - accuracy:
0.9586 - val_loss: 2.2857 - val_accuracy: 0.6350
Epoch 171/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1256 - accuracy:
0.9560 - val_loss: 2.3934 - val_accuracy: 0.6350
Epoch 172/200
80/80 [==============================] - 6s 74ms/step - loss: 0.1095 - accuracy:
0.9622 - val_loss: 2.2504 - val_accuracy: 0.6175
Epoch 173/200
80/80 [==============================] - 6s 76ms/step - loss: 0.1033 - accuracy:
0.9656 - val_loss: 2.4406 - val_accuracy: 0.6065
Epoch 174/200
80/80 [==============================] - 6s 78ms/step - loss: 0.1430 - accuracy:
0.9519 - val_loss: 2.3399 - val_accuracy: 0.6390
Epoch 175/200
80/80 [==============================] - 6s 78ms/step - loss: 0.1249 - accuracy:
0.9575 - val_loss: 2.2198 - val_accuracy: 0.6405
Epoch 176/200
80/80 [==============================] - 6s 81ms/step - loss: 0.1124 - accuracy:
0.9640 - val_loss: 2.4217 - val_accuracy: 0.6320
Epoch 177/200
80/80 [==============================] - 4s 55ms/step - loss: 0.1174 - accuracy:
0.9597 - val_loss: 2.3894 - val_accuracy: 0.6420
Epoch 178/200
80/80 [==============================] - 4s 49ms/step - loss: 0.1155 - accuracy:
0.9588 - val_loss: 2.2464 - val_accuracy: 0.6430
Epoch 179/200
80/80 [==============================] - 4s 51ms/step - loss: 0.1066 - accuracy:
0.9647 - val_loss: 2.2374 - val_accuracy: 0.6250
Epoch 180/200
80/80 [==============================] - 4s 50ms/step - loss: 0.0997 - accuracy:
0.9649 - val_loss: 2.2522 - val_accuracy: 0.6390
```

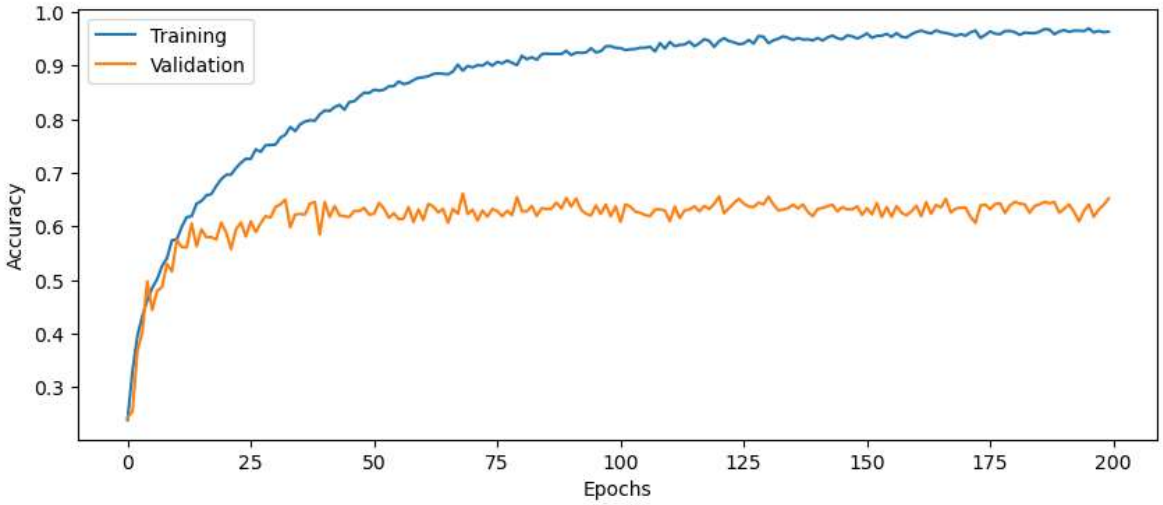
```
Epoch 181/200
80/80 [==============================] - 4s 48ms/step - loss: 0.1102 - accuracy:
0.9625 - val_loss: 2.3959 - val_accuracy: 0.6460
Epoch 182/200
80/80 [==============================] - 4s 49ms/step - loss: 0.1239 - accuracy:
0.9575 - val_loss: 2.2974 - val_accuracy: 0.6425
Epoch 183/200
80/80 [==============================] - 4s 48ms/step - loss: 0.1106 - accuracy:
0.9617 - val_loss: 2.2620 - val_accuracy: 0.6410
Epoch 184/200
80/80 [==============================] - 4s 49ms/step - loss: 0.1116 - accuracy:
0.9609 - val_loss: 2.2849 - val_accuracy: 0.6255
Epoch 185/200
80/80 [==============================] - 4s 49ms/step - loss: 0.1094 - accuracy:
0.9606 - val_loss: 2.2653 - val_accuracy: 0.6380
Epoch 186/200
80/80 [==============================] - 4s 49ms/step - loss: 0.1025 - accuracy:
0.9635 - val_loss: 2.4458 - val_accuracy: 0.6410
Epoch 187/200
80/80 [==============================] - 4s 49ms/step - loss: 0.0971 - accuracy:
0.9682 - val_loss: 2.3013 - val_accuracy: 0.6460
Epoch 188/200
80/80 [==============================] - 4s 48ms/step - loss: 0.1023 - accuracy:
0.9674 - val_loss: 2.3569 - val_accuracy: 0.6430
Epoch 189/200
80/80 [==============================] - 4s 49ms/step - loss: 0.1181 - accuracy:
0.9586 - val_loss: 2.3214 - val_accuracy: 0.6455
Epoch 190/200
80/80 [==============================] - 4s 49ms/step - loss: 0.1051 - accuracy:
0.9638 - val_loss: 2.4162 - val_accuracy: 0.6255
Epoch 191/200
80/80 [==============================] - 4s 48ms/step - loss: 0.1029 - accuracy:
0.9663 - val_loss: 2.3384 - val_accuracy: 0.6330
Epoch 192/200
80/80 [==============================] - 4s 49ms/step - loss: 0.1094 - accuracy:
0.9634 - val_loss: 2.2653 - val_accuracy: 0.6410
Epoch 193/200
80/80 [==============================] - 4s 49ms/step - loss: 0.1030 - accuracy:
0.9653 - val_loss: 2.4494 - val_accuracy: 0.6270
Epoch 194/200
80/80 [==============================] - 4s 49ms/step - loss: 0.1082 - accuracy:
0.9653 - val_loss: 2.4362 - val_accuracy: 0.6095
Epoch 195/200
80/80 [==============================] - 4s 50ms/step - loss: 0.1070 - accuracy:
0.9644 - val_loss: 2.3377 - val_accuracy: 0.6290
Epoch 196/200
80/80 [==============================] - 4s 51ms/step - loss: 0.0945 - accuracy:
0.9696 - val_loss: 2.4905 - val_accuracy: 0.6410
Epoch 197/200
80/80 [==============================] - 4s 50ms/step - loss: 0.1116 - accuracy:
0.9621 - val_loss: 2.5949 - val_accuracy: 0.6180
Epoch 198/200
80/80 [==============================] - 4s 49ms/step - loss: 0.1051 - accuracy:
0.9649 - val_loss: 2.4584 - val_accuracy: 0.6315
Epoch 199/200
80/80 [==============================] - 4s 49ms/step - loss: 0.1090 - accuracy:
0.9625 - val_loss: 2.3621 - val_accuracy: 0.6400
Epoch 200/200
80/80 [==============================] - 4s 49ms/step - loss: 0.1098 - accuracy:
0.9635 - val_loss: 2.1709 - val_accuracy: 0.6520
```

In [ ]:
```
# Check if there is still a big difference in accuracy for original and rotated

# Evaluate the trained model on original test set
score = model6.evaluate(Xtest, Ytest, batch_size = batch_size, verbose=0)
print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])

# Evaluate the trained model on rotated test set
score = model6.evaluate(Xtest_rotated, Ytest, batch_size = batch_size, verbose=0
print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
Test loss: 2.3961
Test accuracy: 0.6185
Test loss: 6.8235
Test accuracy: 0.2560
```

In [ ]:
```
# Plot the history from the training run
plot_results(history6)
```





Part 20: Plot misclassified images

Lets plot some images where the CNN performed badly, these cells are already finished.

In [ ]:
```
# Find misclassified images
y_pred=model6.predict(Xtest)
```

```
y_pred=np.argmax(y_pred,axis=1)

y_correct = np.argmax(Ytest,axis=-1)

miss = np.flatnonzero(y_correct != y_pred)
```

In [ ]:
```
# Plot a few of them
plt.figure(figsize=(15,4))
perm = np.random.permutation(miss)
for i in range(18):
    im = (Xtest[perm[i]] + 1) * 127.5
    im = im.astype('int')
    label_correct = y_correct[perm[i]]
    label_pred = y_pred[perm[i]]

    plt.subplot(3,6,i+1)
    plt.tight_layout()
    plt.imshow(im)
    plt.axis('off')
    plt.title("{}, classified as {}".format(classes[label_correct], classes[labe
plt.show()
```

Part 21: Testing on another size

Question 25: This CNN has been trained on 32 x 32 images, can it be applied to images of another size? If not, why is this the case?

- no, the input parameter will be different(also the flattened fatures) and the trained parameters(e.g. weights of dense layer) wont work at all for a different input dimension

Question 26: Is it possible to design a CNN that can be trained on images of one size, and then applied to an image of any size? How?

- it might be possible via making the convolution/pooling layers dynamic and output the same flattened dimensions

Part 22: Pre-trained 2D CNNs

There are many deep 2D CNNs that have been pre-trained using the large ImageNet database (several million images, 1000 classes). Import a pre-trained ResNet50 network from Keras applications. Show the network using `model.summary()`

Question 27: How many convolutional layers does ResNet50 have?

- 5 convolution layers

Question 28: How many trainable parameters does the ResNet50 network have?

- Trainable params: 25,583,592

Question 29: What is the size of the images that ResNet50 expects as input?

- 224, 224

Question 30: Using the answer to question 28, explain why the second derivative is seldom used when training deep networks.

- it will increase the number of Trainable params multiplicatively, and it will be extra horrofying when we have too much nodes in layers

Apply the pre-trained CNN to 5 random color images that you download and copy to the cloud machine or your own computer. Are the predictions correct? How certain is the network of each image class?

These pre-trained networks can be fine tuned to your specific data, and normally only the last layers need to be re-trained, but it will still be too time consuming to do in this laboration.

See https://keras.io/api/applications/ and https://keras.io/api/applications/resnet/#resnet50-function

Useful functions

`image.load_img` in tensorflow.keras.preprocessing

`image.img_to_array` in tensorflow.keras.preprocessing

`ResNet50` in tensorflow.keras.applications.resnet50

`preprocess_input` in tensorflow.keras.applications.resnet50

`decode_predictions` in tensorflow.keras.applications.resnet50

`expand_dims` in numpy

In [ ]:
```
# Your code for using pre-trained ResNet 50 on 5 color images of your choice.
# The preprocessing should transform the image to a size that is expected by the

from keras.models import Model
from keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_prec
from numpy import expand_dims

model7 = ResNet50(weights='imagenet')
model7.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5
102973440/102967424 [==============================] - 9s 0us/step
Model: "resnet50"
__________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
input_5 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________
conv1_pad (ZeroPadding2D)       (None, 230, 230, 3)  0           input_5[0][0]
__________________________________________________________________________________
conv1_conv (Conv2D)             (None, 112, 112, 64) 9472        conv1_pad[0][0]
__________________________________________________________________________________
conv1_bn (BatchNormalization)   (None, 112, 112, 64) 256         conv1_conv[0][0]
__________________________________________________________________________________
conv1_relu (Activation)         (None, 112, 112, 64) 0           conv1_bn[0][0]
__________________________________________________________________________________
pool1_pad (ZeroPadding2D)       (None, 114, 114, 64) 0           conv1_relu[0][0]
__________________________________________________________________________________
pool1_pool (MaxPooling2D)       (None, 56, 56, 64)   0           pool1_pad[0][0]
__________________________________________________________________________________
conv2_block1_1_conv (Conv2D)    (None, 56, 56, 64)   4160        pool1_pool[0][0]
__________________________________________________________________________________
conv2_block1_1_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block1_1_c
onv[0][0]
__________________________________________________________________________________
conv2_block1_1_relu (Activation (None, 56, 56, 64)   0           conv2_block1_1_b
n[0][0]
__________________________________________________________________________________
conv2_block1_2_conv (Conv2D)    (None, 56, 56, 64)   36928       conv2_block1_1_r
elu[0][0]
__________________________________________________________________________________
conv2_block1_2_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block1_2_c
onv[0][0]
__________________________________________________________________________________
conv2_block1_2_relu (Activation (None, 56, 56, 64)   0           conv2_block1_2_b
n[0][0]
__________________________________________________________________________________
conv2_block1_0_conv (Conv2D)    (None, 56, 56, 256)  16640       pool1_pool[0][0]
__________________________________________________________________________________
conv2_block1_3_conv (Conv2D)    (None, 56, 56, 256)  16640       conv2_block1_2_r
elu[0][0]
__________________________________________________________________________________
```

```
conv2_block1_0_bn (BatchNormali (None, 56, 56, 256)  1024        conv2_block1_0_c
onv[0][0]
__________________________________________________________________________________
__________________
conv2_block1_3_bn (BatchNormali (None, 56, 56, 256)  1024        conv2_block1_3_c
onv[0][0]
__________________________________________________________________________________
__________________
conv2_block1_add (Add)          (None, 56, 56, 256)  0           conv2_block1_0_b
n[0][0]
                                                                 conv2_block1_3_b
n[0][0]
__________________________________________________________________________________
__________________
conv2_block1_out (Activation)   (None, 56, 56, 256)  0           conv2_block1_add
[0][0]
__________________________________________________________________________________
__________________
conv2_block2_1_conv (Conv2D)    (None, 56, 56, 64)   16448       conv2_block1_out
[0][0]
__________________________________________________________________________________
__________________
conv2_block2_1_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block2_1_c
onv[0][0]
__________________________________________________________________________________
__________________
conv2_block2_1_relu (Activation (None, 56, 56, 64)   0           conv2_block2_1_b
n[0][0]
__________________________________________________________________________________
__________________
conv2_block2_2_conv (Conv2D)    (None, 56, 56, 64)   36928       conv2_block2_1_r
elu[0][0]
__________________________________________________________________________________
__________________
conv2_block2_2_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block2_2_c
onv[0][0]
__________________________________________________________________________________
__________________
conv2_block2_2_relu (Activation (None, 56, 56, 64)   0           conv2_block2_2_b
n[0][0]
__________________________________________________________________________________
__________________
conv2_block2_3_conv (Conv2D)    (None, 56, 56, 256)  16640       conv2_block2_2_r
elu[0][0]
__________________________________________________________________________________
__________________
conv2_block2_3_bn (BatchNormali (None, 56, 56, 256)  1024        conv2_block2_3_c
onv[0][0]
__________________________________________________________________________________
__________________
conv2_block2_add (Add)          (None, 56, 56, 256)  0           conv2_block1_out
[0][0]
                                                                 conv2_block2_3_b
n[0][0]
__________________________________________________________________________________
__________________
conv2_block2_out (Activation)   (None, 56, 56, 256)  0           conv2_block2_add
[0][0]
__________________________________________________________________________________
__________________
```

```
conv2_block3_1_conv (Conv2D)    (None, 56, 56, 64)   16448       conv2_block2_out
[0][0]
__________________________________________________________________________________
__________________
conv2_block3_1_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block3_1_c
onv[0][0]
__________________________________________________________________________________
__________________
conv2_block3_1_relu (Activation (None, 56, 56, 64)   0           conv2_block3_1_b
n[0][0]
__________________________________________________________________________________
__________________
conv2_block3_2_conv (Conv2D)    (None, 56, 56, 64)   36928       conv2_block3_1_r
elu[0][0]
__________________________________________________________________________________
__________________
conv2_block3_2_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block3_2_c
onv[0][0]
__________________________________________________________________________________
__________________
conv2_block3_2_relu (Activation (None, 56, 56, 64)   0           conv2_block3_2_b
n[0][0]
__________________________________________________________________________________
__________________
conv2_block3_3_conv (Conv2D)    (None, 56, 56, 256)  16640       conv2_block3_2_r
elu[0][0]
__________________________________________________________________________________
__________________
conv2_block3_3_bn (BatchNormali (None, 56, 56, 256)  1024        conv2_block3_3_c
onv[0][0]
__________________________________________________________________________________
__________________
conv2_block3_add (Add)          (None, 56, 56, 256)  0           conv2_block2_out
[0][0]
                                                                 conv2_block3_3_b
n[0][0]
__________________________________________________________________________________
__________________
conv2_block3_out (Activation)   (None, 56, 56, 256)  0           conv2_block3_add
[0][0]
__________________________________________________________________________________
__________________
conv3_block1_1_conv (Conv2D)    (None, 28, 28, 128)  32896       conv2_block3_out
[0][0]
__________________________________________________________________________________
__________________
conv3_block1_1_bn (BatchNormali (None, 28, 28, 128)  512         conv3_block1_1_c
onv[0][0]
__________________________________________________________________________________
__________________
conv3_block1_1_relu (Activation (None, 28, 28, 128)  0           conv3_block1_1_b
n[0][0]
__________________________________________________________________________________
__________________
conv3_block1_2_conv (Conv2D)    (None, 28, 28, 128)  147584      conv3_block1_1_r
elu[0][0]
__________________________________________________________________________________
__________________
conv3_block1_2_bn (BatchNormali (None, 28, 28, 128)  512         conv3_block1_2_c
onv[0][0]
```

```
__________________________________________________________________________________________________
conv3_block1_2_relu (Activation (None, 28, 28, 128)  0           conv3_block1_2_b
n[0][0]
__________________________________________________________________________________________________
conv3_block1_0_conv (Conv2D)    (None, 28, 28, 512)  131584      conv2_block3_out
[0][0]
__________________________________________________________________________________________________
conv3_block1_3_conv (Conv2D)    (None, 28, 28, 512)  66048       conv3_block1_2_r
elu[0][0]
__________________________________________________________________________________________________
conv3_block1_0_bn (BatchNormali (None, 28, 28, 512)  2048        conv3_block1_0_c
onv[0][0]
__________________________________________________________________________________________________
conv3_block1_3_bn (BatchNormali (None, 28, 28, 512)  2048        conv3_block1_3_c
onv[0][0]
__________________________________________________________________________________________________
conv3_block1_add (Add)          (None, 28, 28, 512)  0           conv3_block1_0_b
n[0][0]
                                                                 conv3_block1_3_b
n[0][0]
__________________________________________________________________________________________________
conv3_block1_out (Activation)   (None, 28, 28, 512)  0           conv3_block1_add
[0][0]
__________________________________________________________________________________________________
conv3_block2_1_conv (Conv2D)    (None, 28, 28, 128)  65664       conv3_block1_out
[0][0]
__________________________________________________________________________________________________
conv3_block2_1_bn (BatchNormali (None, 28, 28, 128)  512         conv3_block2_1_c
onv[0][0]
__________________________________________________________________________________________________
conv3_block2_1_relu (Activation (None, 28, 28, 128)  0           conv3_block2_1_b
n[0][0]
__________________________________________________________________________________________________
conv3_block2_2_conv (Conv2D)    (None, 28, 28, 128)  147584      conv3_block2_1_r
elu[0][0]
__________________________________________________________________________________________________
conv3_block2_2_bn (BatchNormali (None, 28, 28, 128)  512         conv3_block2_2_c
onv[0][0]
__________________________________________________________________________________________________
conv3_block2_2_relu (Activation (None, 28, 28, 128)  0           conv3_block2_2_b
n[0][0]
__________________________________________________________________________________________________
conv3_block2_3_conv (Conv2D)    (None, 28, 28, 512)  66048       conv3_block2_2_r
elu[0][0]
__________________________________________________________________________________________________
```

```
conv3_block2_3_bn (BatchNormali (None, 28, 28, 512)  2048        conv3_block2_3_c
onv[0][0]
__________________________________________________________________________________
__________________
conv3_block2_add (Add)          (None, 28, 28, 512)  0           conv3_block1_out
[0][0]
                                                                 conv3_block2_3_b
n[0][0]
__________________________________________________________________________________
__________________
conv3_block2_out (Activation)   (None, 28, 28, 512)  0           conv3_block2_add
[0][0]
__________________________________________________________________________________
__________________
conv3_block3_1_conv (Conv2D)    (None, 28, 28, 128)  65664       conv3_block2_out
[0][0]
__________________________________________________________________________________
__________________
conv3_block3_1_bn (BatchNormali (None, 28, 28, 128)  512         conv3_block3_1_c
onv[0][0]
__________________________________________________________________________________
__________________
conv3_block3_1_relu (Activation (None, 28, 28, 128)  0           conv3_block3_1_b
n[0][0]
__________________________________________________________________________________
__________________
conv3_block3_2_conv (Conv2D)    (None, 28, 28, 128)  147584      conv3_block3_1_r
elu[0][0]
__________________________________________________________________________________
__________________
conv3_block3_2_bn (BatchNormali (None, 28, 28, 128)  512         conv3_block3_2_c
onv[0][0]
__________________________________________________________________________________
__________________
conv3_block3_2_relu (Activation (None, 28, 28, 128)  0           conv3_block3_2_b
n[0][0]
__________________________________________________________________________________
__________________
conv3_block3_3_conv (Conv2D)    (None, 28, 28, 512)  66048       conv3_block3_2_r
elu[0][0]
__________________________________________________________________________________
__________________
conv3_block3_3_bn (BatchNormali (None, 28, 28, 512)  2048        conv3_block3_3_c
onv[0][0]
__________________________________________________________________________________
__________________
conv3_block3_add (Add)          (None, 28, 28, 512)  0           conv3_block2_out
[0][0]
                                                                 conv3_block3_3_b
n[0][0]
__________________________________________________________________________________
__________________
conv3_block3_out (Activation)   (None, 28, 28, 512)  0           conv3_block3_add
[0][0]
__________________________________________________________________________________
__________________
conv3_block4_1_conv (Conv2D)    (None, 28, 28, 128)  65664       conv3_block3_out
[0][0]
__________________________________________________________________________________
__________________
```

```
conv3_block4_1_bn (BatchNormali (None, 28, 28, 128)  512         conv3_block4_1_c
onv[0][0]
__________________________________________________________________________________
__________________
conv3_block4_1_relu (Activation (None, 28, 28, 128)  0           conv3_block4_1_b
n[0][0]
__________________________________________________________________________________
__________________
conv3_block4_2_conv (Conv2D)    (None, 28, 28, 128)  147584      conv3_block4_1_r
elu[0][0]
__________________________________________________________________________________
__________________
conv3_block4_2_bn (BatchNormali (None, 28, 28, 128)  512         conv3_block4_2_c
onv[0][0]
__________________________________________________________________________________
__________________
conv3_block4_2_relu (Activation (None, 28, 28, 128)  0           conv3_block4_2_b
n[0][0]
__________________________________________________________________________________
__________________
conv3_block4_3_conv (Conv2D)    (None, 28, 28, 512)  66048       conv3_block4_2_r
elu[0][0]
__________________________________________________________________________________
__________________
conv3_block4_3_bn (BatchNormali (None, 28, 28, 512)  2048        conv3_block4_3_c
onv[0][0]
__________________________________________________________________________________
__________________
conv3_block4_add (Add)          (None, 28, 28, 512)  0           conv3_block3_out
[0][0]
                                                                 conv3_block4_3_b
n[0][0]
__________________________________________________________________________________
__________________
conv3_block4_out (Activation)   (None, 28, 28, 512)  0           conv3_block4_add
[0][0]
__________________________________________________________________________________
__________________
conv4_block1_1_conv (Conv2D)    (None, 14, 14, 256)  131328      conv3_block4_out
[0][0]
__________________________________________________________________________________
__________________
conv4_block1_1_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block1_1_c
onv[0][0]
__________________________________________________________________________________
__________________
conv4_block1_1_relu (Activation (None, 14, 14, 256)  0           conv4_block1_1_b
n[0][0]
__________________________________________________________________________________
__________________
conv4_block1_2_conv (Conv2D)    (None, 14, 14, 256)  590080      conv4_block1_1_r
elu[0][0]
__________________________________________________________________________________
__________________
conv4_block1_2_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block1_2_c
onv[0][0]
__________________________________________________________________________________
__________________
conv4_block1_2_relu (Activation (None, 14, 14, 256)  0           conv4_block1_2_b
n[0][0]
```

```
__________________________________________________________________________________________________
conv4_block1_0_conv (Conv2D)    (None, 14, 14, 1024) 525312      conv3_block4_out
[0][0]
__________________________________________________________________________________________________
conv4_block1_3_conv (Conv2D)    (None, 14, 14, 1024) 263168      conv4_block1_2_r
elu[0][0]
__________________________________________________________________________________________________
conv4_block1_0_bn (BatchNormali (None, 14, 14, 1024) 4096        conv4_block1_0_c
onv[0][0]
__________________________________________________________________________________________________
conv4_block1_3_bn (BatchNormali (None, 14, 14, 1024) 4096        conv4_block1_3_c
onv[0][0]
__________________________________________________________________________________________________
conv4_block1_add (Add)          (None, 14, 14, 1024) 0           conv4_block1_0_b
n[0][0]
                                                                 conv4_block1_3_b
n[0][0]
__________________________________________________________________________________________________
conv4_block1_out (Activation)   (None, 14, 14, 1024) 0           conv4_block1_add
[0][0]
__________________________________________________________________________________________________
conv4_block2_1_conv (Conv2D)    (None, 14, 14, 256)  262400      conv4_block1_out
[0][0]
__________________________________________________________________________________________________
conv4_block2_1_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block2_1_c
onv[0][0]
__________________________________________________________________________________________________
conv4_block2_1_relu (Activation (None, 14, 14, 256)  0           conv4_block2_1_b
n[0][0]
__________________________________________________________________________________________________
conv4_block2_2_conv (Conv2D)    (None, 14, 14, 256)  590080      conv4_block2_1_r
elu[0][0]
__________________________________________________________________________________________________
conv4_block2_2_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block2_2_c
onv[0][0]
__________________________________________________________________________________________________
conv4_block2_2_relu (Activation (None, 14, 14, 256)  0           conv4_block2_2_b
n[0][0]
__________________________________________________________________________________________________
conv4_block2_3_conv (Conv2D)    (None, 14, 14, 1024) 263168      conv4_block2_2_r
elu[0][0]
__________________________________________________________________________________________________
conv4_block2_3_bn (BatchNormali (None, 14, 14, 1024) 4096        conv4_block2_3_c
onv[0][0]
__________________________________________________________________________________________________
```

```
conv4_block2_add (Add)          (None, 14, 14, 1024) 0           conv4_block1_out
[0][0]
                                                                 conv4_block2_3_b
n[0][0]
__________________________________________________________________________________
_________________
conv4_block2_out (Activation)   (None, 14, 14, 1024) 0           conv4_block2_add
[0][0]
__________________________________________________________________________________
_________________
conv4_block3_1_conv (Conv2D)    (None, 14, 14, 256)  262400      conv4_block2_out
[0][0]
__________________________________________________________________________________
_________________
conv4_block3_1_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block3_1_c
onv[0][0]
__________________________________________________________________________________
_________________
conv4_block3_1_relu (Activation (None, 14, 14, 256)  0           conv4_block3_1_b
n[0][0]
__________________________________________________________________________________
_________________
conv4_block3_2_conv (Conv2D)    (None, 14, 14, 256)  590080      conv4_block3_1_r
elu[0][0]
__________________________________________________________________________________
_________________
conv4_block3_2_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block3_2_c
onv[0][0]
__________________________________________________________________________________
_________________
conv4_block3_2_relu (Activation (None, 14, 14, 256)  0           conv4_block3_2_b
n[0][0]
__________________________________________________________________________________
_________________
conv4_block3_3_conv (Conv2D)    (None, 14, 14, 1024) 263168      conv4_block3_2_r
elu[0][0]
__________________________________________________________________________________
_________________
conv4_block3_3_bn (BatchNormali (None, 14, 14, 1024) 4096        conv4_block3_3_c
onv[0][0]
__________________________________________________________________________________
_________________
conv4_block3_add (Add)          (None, 14, 14, 1024) 0           conv4_block2_out
[0][0]
                                                                 conv4_block3_3_b
n[0][0]
__________________________________________________________________________________
_________________
conv4_block3_out (Activation)   (None, 14, 14, 1024) 0           conv4_block3_add
[0][0]
__________________________________________________________________________________
_________________
conv4_block4_1_conv (Conv2D)    (None, 14, 14, 256)  262400      conv4_block3_out
[0][0]
__________________________________________________________________________________
_________________
conv4_block4_1_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block4_1_c
onv[0][0]
__________________________________________________________________________________
_________________
```

```
conv4_block4_1_relu (Activation (None, 14, 14, 256)  0           conv4_block4_1_b
n[0][0]
__________________________________________________________________________________
__________________
conv4_block4_2_conv (Conv2D)    (None, 14, 14, 256)  590080      conv4_block4_1_r
elu[0][0]
__________________________________________________________________________________
__________________
conv4_block4_2_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block4_2_c
onv[0][0]
__________________________________________________________________________________
__________________
conv4_block4_2_relu (Activation (None, 14, 14, 256)  0           conv4_block4_2_b
n[0][0]
__________________________________________________________________________________
__________________
conv4_block4_3_conv (Conv2D)    (None, 14, 14, 1024) 263168      conv4_block4_2_r
elu[0][0]
__________________________________________________________________________________
__________________
conv4_block4_3_bn (BatchNormali (None, 14, 14, 1024) 4096        conv4_block4_3_c
onv[0][0]
__________________________________________________________________________________
__________________
conv4_block4_add (Add)          (None, 14, 14, 1024) 0           conv4_block3_out
[0][0]
                                                                 conv4_block4_3_b
n[0][0]
__________________________________________________________________________________
__________________
conv4_block4_out (Activation)   (None, 14, 14, 1024) 0           conv4_block4_add
[0][0]
__________________________________________________________________________________
__________________
conv4_block5_1_conv (Conv2D)    (None, 14, 14, 256)  262400      conv4_block4_out
[0][0]
__________________________________________________________________________________
__________________
conv4_block5_1_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block5_1_c
onv[0][0]
__________________________________________________________________________________
__________________
conv4_block5_1_relu (Activation (None, 14, 14, 256)  0           conv4_block5_1_b
n[0][0]
__________________________________________________________________________________
__________________
conv4_block5_2_conv (Conv2D)    (None, 14, 14, 256)  590080      conv4_block5_1_r
elu[0][0]
__________________________________________________________________________________
__________________
conv4_block5_2_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block5_2_c
onv[0][0]
__________________________________________________________________________________
__________________
conv4_block5_2_relu (Activation (None, 14, 14, 256)  0           conv4_block5_2_b
n[0][0]
__________________________________________________________________________________
__________________
conv4_block5_3_conv (Conv2D)    (None, 14, 14, 1024) 263168      conv4_block5_2_r
elu[0][0]
```

```
__________________________________________________________________________________________________
conv4_block5_3_bn (BatchNormali (None, 14, 14, 1024) 4096        conv4_block5_3_c
onv[0][0]
__________________________________________________________________________________________________
conv4_block5_add (Add)          (None, 14, 14, 1024) 0           conv4_block4_out
[0][0]
                                                                 conv4_block5_3_b
n[0][0]
__________________________________________________________________________________________________
conv4_block5_out (Activation)   (None, 14, 14, 1024) 0           conv4_block5_add
[0][0]
__________________________________________________________________________________________________
conv4_block6_1_conv (Conv2D)    (None, 14, 14, 256)  262400      conv4_block5_out
[0][0]
__________________________________________________________________________________________________
conv4_block6_1_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block6_1_c
onv[0][0]
__________________________________________________________________________________________________
conv4_block6_1_relu (Activation (None, 14, 14, 256)  0           conv4_block6_1_b
n[0][0]
__________________________________________________________________________________________________
conv4_block6_2_conv (Conv2D)    (None, 14, 14, 256)  590080      conv4_block6_1_r
elu[0][0]
__________________________________________________________________________________________________
conv4_block6_2_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block6_2_c
onv[0][0]
__________________________________________________________________________________________________
conv4_block6_2_relu (Activation (None, 14, 14, 256)  0           conv4_block6_2_b
n[0][0]
__________________________________________________________________________________________________
conv4_block6_3_conv (Conv2D)    (None, 14, 14, 1024) 263168      conv4_block6_2_r
elu[0][0]
__________________________________________________________________________________________________
conv4_block6_3_bn (BatchNormali (None, 14, 14, 1024) 4096        conv4_block6_3_c
onv[0][0]
__________________________________________________________________________________________________
conv4_block6_add (Add)          (None, 14, 14, 1024) 0           conv4_block5_out
[0][0]
                                                                 conv4_block6_3_b
n[0][0]
__________________________________________________________________________________________________
conv4_block6_out (Activation)   (None, 14, 14, 1024) 0           conv4_block6_add
[0][0]
__________________________________________________________________________________________________
conv5_block1_1_conv (Conv2D)    (None, 7, 7, 512)    524800      conv4_block6_out
[0][0]
```

```
__________________________________________________________________________________________________
conv5_block1_1_bn (BatchNormali (None, 7, 7, 512)    2048        conv5_block1_1_c
onv[0][0]
__________________________________________________________________________________________________
conv5_block1_1_relu (Activation (None, 7, 7, 512)    0           conv5_block1_1_b
n[0][0]
__________________________________________________________________________________________________
conv5_block1_2_conv (Conv2D)    (None, 7, 7, 512)    2359808     conv5_block1_1_r
elu[0][0]
__________________________________________________________________________________________________
conv5_block1_2_bn (BatchNormali (None, 7, 7, 512)    2048        conv5_block1_2_c
onv[0][0]
__________________________________________________________________________________________________
conv5_block1_2_relu (Activation (None, 7, 7, 512)    0           conv5_block1_2_b
n[0][0]
__________________________________________________________________________________________________
conv5_block1_0_conv (Conv2D)    (None, 7, 7, 2048)   2099200     conv4_block6_out
[0][0]
__________________________________________________________________________________________________
conv5_block1_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624     conv5_block1_2_r
elu[0][0]
__________________________________________________________________________________________________
conv5_block1_0_bn (BatchNormali (None, 7, 7, 2048)   8192        conv5_block1_0_c
onv[0][0]
__________________________________________________________________________________________________
conv5_block1_3_bn (BatchNormali (None, 7, 7, 2048)   8192        conv5_block1_3_c
onv[0][0]
__________________________________________________________________________________________________
conv5_block1_add (Add)          (None, 7, 7, 2048)   0           conv5_block1_0_b
n[0][0]
                                                                 conv5_block1_3_b
n[0][0]
__________________________________________________________________________________________________
conv5_block1_out (Activation)   (None, 7, 7, 2048)   0           conv5_block1_add
[0][0]
__________________________________________________________________________________________________
conv5_block2_1_conv (Conv2D)    (None, 7, 7, 512)    1049088     conv5_block1_out
[0][0]
__________________________________________________________________________________________________
conv5_block2_1_bn (BatchNormali (None, 7, 7, 512)    2048        conv5_block2_1_c
onv[0][0]
__________________________________________________________________________________________________
conv5_block2_1_relu (Activation (None, 7, 7, 512)    0           conv5_block2_1_b
n[0][0]
__________________________________________________________________________________________________
```

```
conv5_block2_2_conv (Conv2D)    (None, 7, 7, 512)    2359808     conv5_block2_1_r
elu[0][0]
__________________________________________________________________________________
__________________
conv5_block2_2_bn (BatchNormali (None, 7, 7, 512)    2048        conv5_block2_2_c
onv[0][0]
__________________________________________________________________________________
__________________
conv5_block2_2_relu (Activation (None, 7, 7, 512)    0           conv5_block2_2_b
n[0][0]
__________________________________________________________________________________
__________________
conv5_block2_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624     conv5_block2_2_r
elu[0][0]
__________________________________________________________________________________
__________________
conv5_block2_3_bn (BatchNormali (None, 7, 7, 2048)   8192        conv5_block2_3_c
onv[0][0]
__________________________________________________________________________________
__________________
conv5_block2_add (Add)          (None, 7, 7, 2048)   0           conv5_block1_out
[0][0]
                                                                 conv5_block2_3_b
n[0][0]
__________________________________________________________________________________
__________________
conv5_block2_out (Activation)   (None, 7, 7, 2048)   0           conv5_block2_add
[0][0]
__________________________________________________________________________________
__________________
conv5_block3_1_conv (Conv2D)    (None, 7, 7, 512)    1049088     conv5_block2_out
[0][0]
__________________________________________________________________________________
__________________
conv5_block3_1_bn (BatchNormali (None, 7, 7, 512)    2048        conv5_block3_1_c
onv[0][0]
__________________________________________________________________________________
__________________
conv5_block3_1_relu (Activation (None, 7, 7, 512)    0           conv5_block3_1_b
n[0][0]
__________________________________________________________________________________
__________________
conv5_block3_2_conv (Conv2D)    (None, 7, 7, 512)    2359808     conv5_block3_1_r
elu[0][0]
__________________________________________________________________________________
__________________
conv5_block3_2_bn (BatchNormali (None, 7, 7, 512)    2048        conv5_block3_2_c
onv[0][0]
__________________________________________________________________________________
__________________
conv5_block3_2_relu (Activation (None, 7, 7, 512)    0           conv5_block3_2_b
n[0][0]
__________________________________________________________________________________
__________________
conv5_block3_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624     conv5_block3_2_r
elu[0][0]
__________________________________________________________________________________
__________________
conv5_block3_3_bn (BatchNormali (None, 7, 7, 2048)   8192        conv5_block3_3_c
onv[0][0]
```

```
__________________________________________________________________________________________________
conv5_block3_add (Add)          (None, 7, 7, 2048)   0           conv5_block2_out
[0][0]
                                                                 conv5_block3_3_b
n[0][0]
__________________________________________________________________________________________________
conv5_block3_out (Activation)   (None, 7, 7, 2048)   0           conv5_block3_add
[0][0]
__________________________________________________________________________________________________
avg_pool (GlobalAveragePooling2 (None, 2048)         0           conv5_block3_out
[0][0]
__________________________________________________________________________________________________
predictions (Dense)             (None, 1000)         2049000     avg_pool[0][0]
==================================================================================================
Total params: 25,636,712
Trainable params: 25,583,592
Non-trainable params: 53,120
__________________________________________________________________________________________________
```

In [ ]:
```
img_path = r'C:\Users\ni\Pictures\imagenet\val\ILSVRC2012_val_00000005.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model7.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/dat
a/imagenet_class_index.json
40960/35363 [==================================] - 0s 0us/step
Predicted: [('n03131574', 'crib', 0.3695707), ('n03125729', 'cradle', 0.3031985
5), ('n02804414', 'bassinet', 0.28942806)]
40960/35363 [==================================] - 0s 0us/step
Predicted: [('n03131574', 'crib', 0.3695707), ('n03125729', 'cradle', 0.3031985
5), ('n02804414', 'bassinet', 0.28942806)]
```

In [ ]:
```
img_path = r"C:\Users\ni\Pictures\imagenet\val\ILSVRC2012_val_00000004.jpg"
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model7.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

```
Predicted: [('n07920052', 'espresso', 0.52577883), ('n07930864', 'cup', 0.2248913
6), ('n03063599', 'coffee_mug', 0.13086708)]
```

In [ ]:
```
img_path = r'C:\Users\ni\Pictures\imagenet\val\ILSVRC2012_val_00000003.jpg'
img = image.load_img(img_path, target_size=(224, 224))
```

```
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model7.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

```
Predicted: [('n02106030', 'collie', 0.98700887), ('n02105855', 'Shetland_sheepdo
g', 0.0119160265), ('n02090622', 'borzoi', 0.0008067721)]
```

In [ ]:
```
img_path = r'C:\Users\ni\Pictures\imagenet\val\ILSVRC2012_val_00000002.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model7.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

```
Predicted: [('n04228054', 'ski', 0.83161676), ('n04208210', 'shovel', 0.1404188
7), ('n09193705', 'alp', 0.024471473)]
```

In [ ]:
```
img_path = r'C:\Users\ni\Pictures\imagenet\val\ILSVRC2012_val_00000001.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model7.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

```
Predicted: [('n01751748', 'sea_snake', 0.6656197), ('n01729322', 'hognose_snake',
0.26146558), ('n01737021', 'water_snake', 0.03216641)]
```