gravitational_wave_recognition_with_deep_and_transfer_learning_col

January 5, 2021

Gravitational Wave Classification using Convolutional Neural Networks (CNNs)

Introduction

Gravitational Waves (GWs) have recently been observed from binary coalescences of condensed objects (black holes, BHs; neutron stars, NSs). The observations are inherently noisy, with many local noise sources. Thus, a coincident detection approach is useful. Currently, three detectors are in operation: LIGO's (Laser Interferometer Gravitational-Wave Observatory) Hanford and Louisiana detectors in the US, and the Virgo detector in Italy. Here, we present data from the two LIGO detectors. Thus, the data can be assembled in the form of two channels, one from each LIGO detector. In addition, the data, in the form of strain (on the order of fractions of an atomic nuclues, i.e., fractions of a femtometer, 1e-15 m) in the 2 km arms of the laser interferometer detectors, is recorded as a function of frequency (39 frequencies) and time (100 time intervals). Thus, the data can naturally be assembled in the form of a tensor object of size (data_measurments, frequencies=39, time_intervals=100, detector_channels=2). For weak signals (i.e. distant or relatively small BH or NS mass objects), it is not obvious if the data indicate an event or just noise. We present here data consisting of 1568 gravitational wave (GW) events and 1216 nondetection (ND) events (total: 2784), a binary classification problem. The data will be randomized and split as 15-15-70% for test, validation, and training.

Why are we so concerned with quickly determining if a potential event is a true GW event? The reason is that some of these events may also produce radiation which quickly fades. Thus, to further advance the understanding of such extreme astrophysical phenomenon, it is essential to quickly identify true positive GWs and point other instruments (e.g., radio, visible, UV, and x-ray telescopes) in the appropriate direction to observe fast events following the BH/NS collisions.

In the initial part of this project, we have followed the approach and use the data of Michel Kana (see https://github.com/michelkana/Deep-learning-projects), as also described in a Medium article (https://medium.com/swlh/gravitational-waves-explained-83ce37617ab2) and a Toward Data Science article (https://towardsdatascience.com/2020-how-a-i-could-help-astronomers-sorting-big-data-811571705707). We both intially implement a CIFAR-10 type of CNN classifier (with dropout). Whereas Kana stops at 60 epochs, we continue to 75. However, this still does not stabilize the validation data curves with epoch (accuracy or loss), and neither does use of a learning rate smaller by a factor of ten. Thus, we extend past Kana's approach and use transfer learning with a VGG-16 base and a trained binary classifier, which is shown to give exceptionally good performance with minimal effort, despite the rather different types of data introduced here compared to what was used to train VGG-16. This again confirms the power of transfer learning and its applicability across diverse fields.

Data overview

[157]:
```
from google.colab import drive
drive.mount('/content/drive')

import sys
sys.path.append('/content/drive/My Drive/dlpython/Project3')
```

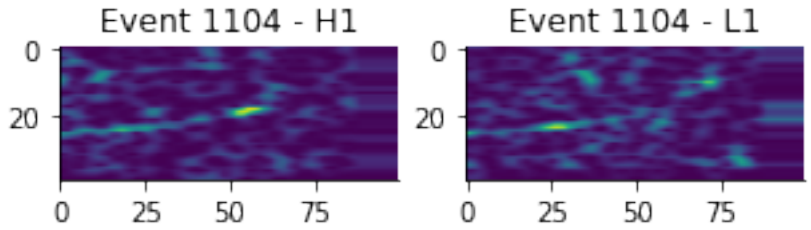
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

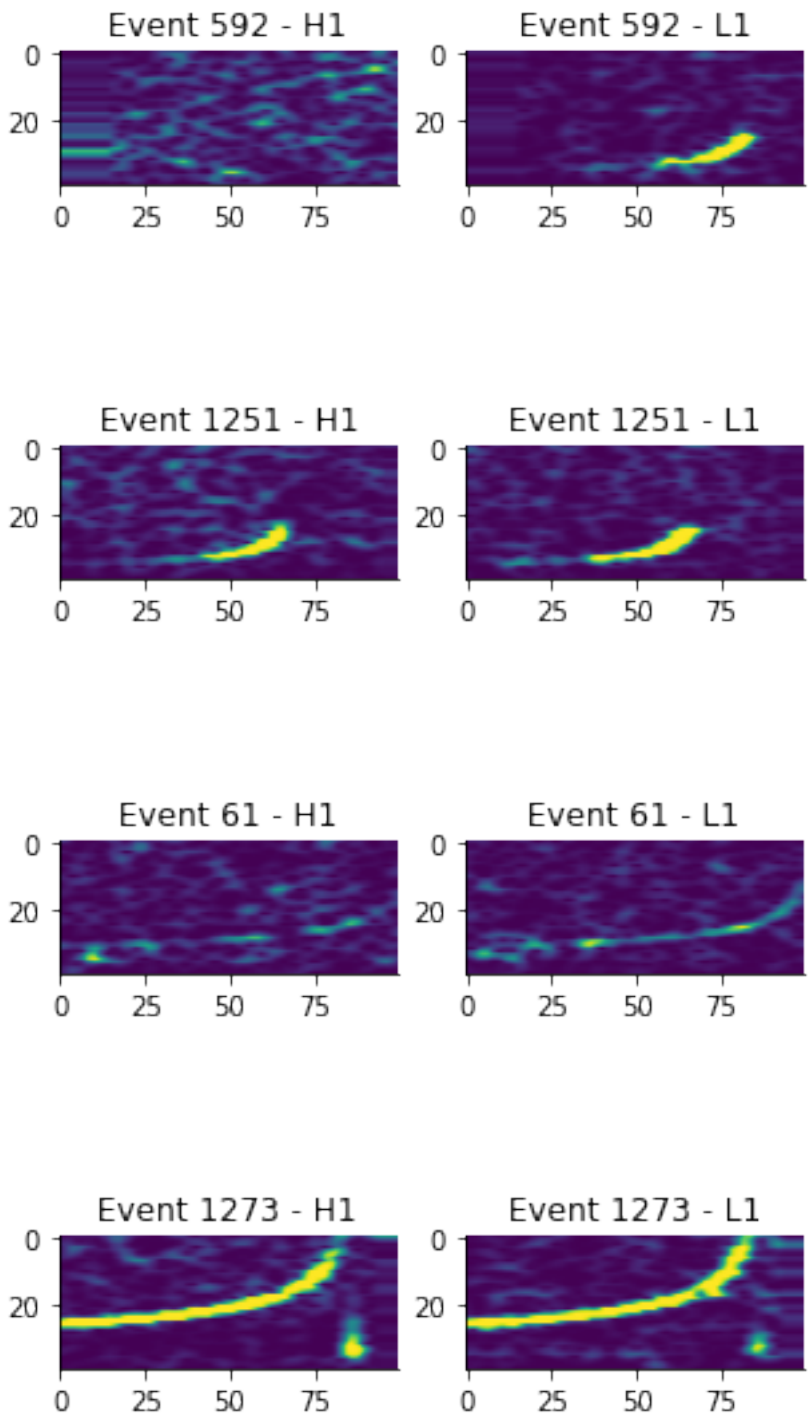
[158]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
from random import seed
from random import randint
import seaborn as sns

seed(30)

GW = np.load('/content/drive/My Drive/dlpython/Project3/data/GW_aug.npy')
ND = np.load('/content/drive/My Drive/dlpython/Project3/data/ND_aug.npy')
GW.shape, ND.shape
```

[158]:
```
((1568, 39, 100, 2), (1216, 39, 100, 2))
```

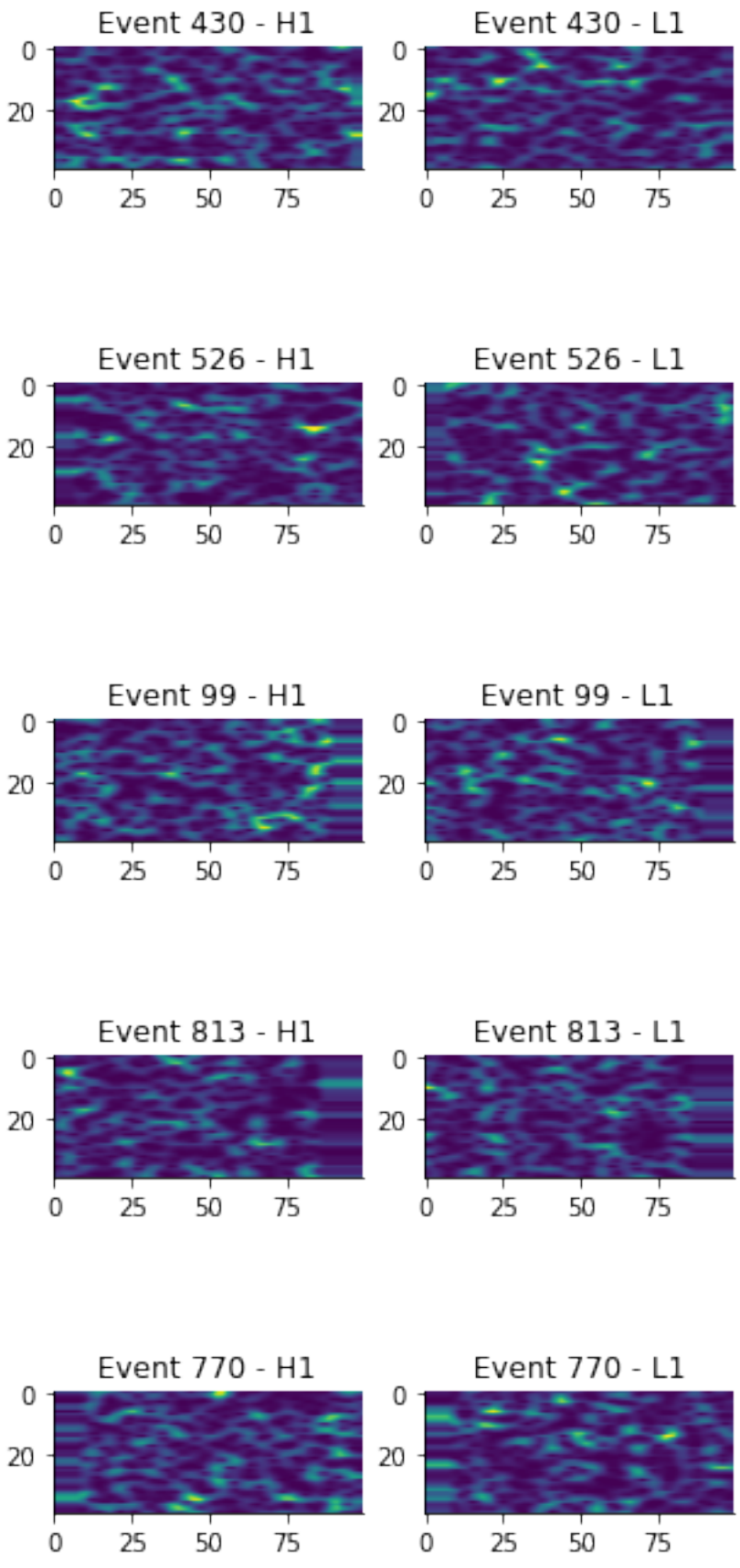
[159]:
```
def plot_event(event=0, data=GW):
    fig, ax = plt.subplots(1,2,figsize=(5,5))
    ax[0].imshow(data[event,:,:,0])
    ax[0].set_title('Event {} - H1'.format(event))
    ax[1].imshow(data[event,:,:,1])
    ax[1].set_title('Event {} - L1'.format(event));
```

[160]:
```
for e in range(5):
    e = randint(0, GW.shape[0])
    plot_event(e)
```



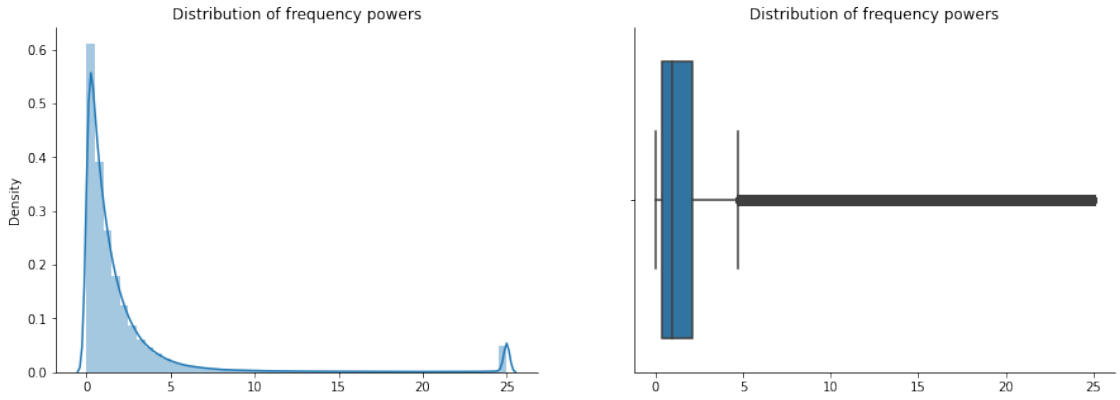


[161]:
```
for e in range(5):
    e = randint(0, ND.shape[0])
    plot_event(e, data=ND)
```


Event 430 - H1
Event 430 - L1
Event 526 - H1
Event 526 - L1
Event 99 - H1
Event 99 - L1
Event 813 - H1
Event 813 - L1
Event 770 - H1
Event 770 - L1
0
20
0
25
50
75

[162]:
```
fig, ax = plt.subplots(1,2,figsize=(15,5))
sns.distplot(GW.flatten(), ax=ax[0])
ax[0].set_title('Distribution of frequency powers')
sns.boxplot(np.unique(GW), ax=ax[1])
ax[1].set_title('Distribution of frequency powers');
```

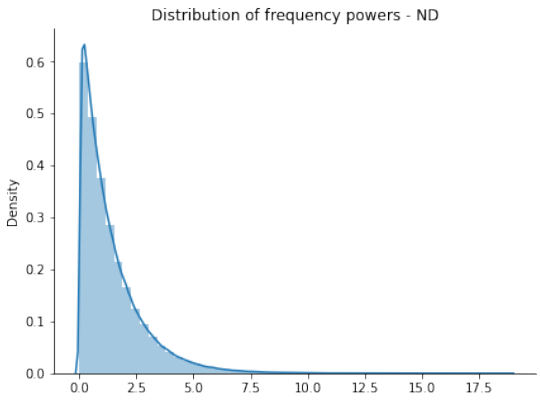
```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

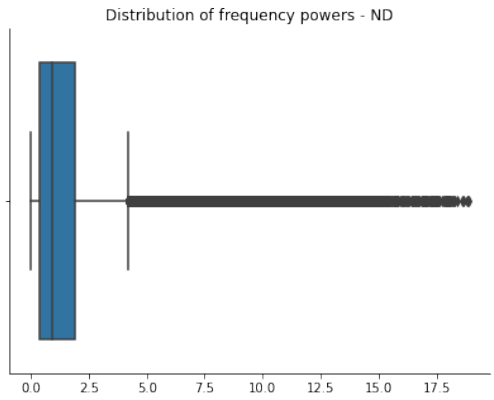


[163]:
```
fig, ax = plt.subplots(1,2,figsize=(15,5))
sns.distplot(ND.flatten(), ax=ax[0])
ax[0].set_title('Distribution of frequency powers - ND')
sns.boxplot(np.unique(ND), ax=ax[1])
ax[1].set_title('Distribution of frequency powers - ND');
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarning:
```

```
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  FutureWarning
```





It is seen that the data are broadly similar. There are some outliers in the true GW events.

Data preprocessing and splitting

Now we normalize and split the data.

[164]:
```
GW = GW.astype('float32')/np.max(GW)
ND = ND.astype('float32')/np.max(ND)

np.random.seed(30)
np.random.shuffle(GW)
np.random.shuffle(ND)
```

[165]:
```
X = np.append(GW, ND, axis=0)
Y = np.append(np.ones((GW.shape[0],1)), np.zeros((ND.shape[0],1)), axis=0)
X.shape, Y.shape, GW.shape, ND.shape
```

[165]:
```
((2784, 39, 100, 2), (2784, 1), (1568, 39, 100, 2), (1216, 39, 100, 2))
```

[166]:
```
test_val_idx = np.random.choice(np.arange(0, Y.shape[0], 1), size=round(.3*Y.
↪shape[0]), replace=False)
```

[167]:
```
test_idx = np.random.choice(test_val_idx, size=round(.5*test_val_idx.shape[0]),␣
↪replace=False)
```

[168]:
```
val_idx = np.setdiff1d(test_val_idx, test_idx)
```

[169]:
```
train_idx = np.setdiff1d(np.arange(0, Y.shape[0], 1), test_val_idx)
```

[170]:
```
test_idx.shape, val_idx.shape, train_idx.shape
```

[170]:
```
((418,), (417,), (1949,))
```

[171]:
```
test_idx.shape[0]+val_idx.shape[0]+train_idx.shape[0]
```

[171]:
```
2784
```

[172]:
```
np.intersect1d(test_idx, val_idx)
```

[172]:
```
array([], dtype=int64)
```

[173]:
```
np.intersect1d(val_idx, train_idx)
```

[173]:
```
array([], dtype=int64)
```

[174]:
```
np.intersect1d(test_idx, train_idx)
```

[174]:
```
array([], dtype=int64)
```

[175]:
```
print(np.amin(test_idx), np.amax(test_idx))
```

```
8 2775
```

[176]:
```
print(np.amin(val_idx), np.amax(val_idx))
```

```
1 2745
```

[177]:
```
print(np.amin(train_idx), np.amax(train_idx))
```

```
0 2783
```

[178]:
```
X_test = X[test_idx]
y_test = Y[test_idx]
X_val = X[val_idx]
y_val = Y[val_idx]
X_train = X[train_idx]
y_train = Y[train_idx]
```

[179]:
```
print(X_test.shape, y_test.shape, X_val.shape, y_val.shape, X_train.shape,␣
↪y_train.shape)
```

```
(418, 39, 100, 2) (418, 1) (417, 39, 100, 2) (417, 1) (1949, 39, 100, 2) (1949,
1)
```

[180]:
```
np.unique(y_test), np.unique(y_val), np.unique(y_train)
```

[180]:
```
(array([0., 1.]), array([0., 1.]), array([0., 1.]))
```

Methods and Results

1. Construct the CNN: First replicate Kana's CIFAR-10 - style CNN

[181]:
```
#  tensorflow library provides functions for deep neural networks
import tensorflow as tf
# for reproducibility
from numpy.random import seed
seed(30)
# tf.random.set_seed(30)
```

```
#  to get a text report showing the main classification metrics for each class
from sklearn.metrics import classification_report
```

[182]:
```
def create_cnn(input_shape=X_train.shape[1:], nb_classes=1, nb_blocks=3,␣
↪nb_filters=32, filter_size=(3,3),
              pool_size=(2,2), weight_decay=1e-4, padding='same', dropout=.2,␣
↪output_activation='softmax'):

    model = tf.keras.models.Sequential()

    for i in range(nb_blocks):
        if i==0:
            model.add(tf.keras.layers.Conv2D(nb_filters, filter_size,␣
↪activation='relu', padding=padding, kernel_regularizer=tf.keras.regularizers.
↪l2(weight_decay), input_shape=input_shape))
        else:
            model.add(tf.keras.layers.Conv2D(nb_filters, filter_size,␣
↪activation='relu', padding=padding, kernel_regularizer=tf.keras.regularizers.
↪l2(weight_decay)))
        model.add(tf.keras.layers.BatchNormalization())
        model.add(tf.keras.layers.Conv2D(nb_filters, filter_size,␣
↪activation='relu', padding=padding, kernel_regularizer=tf.keras.regularizers.
↪l2(weight_decay)))
        model.add(tf.keras.layers.BatchNormalization())
        model.add(tf.keras.layers.MaxPooling2D(pool_size=pool_size))
        model.add(tf.keras.layers.Dropout(dropout))

    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(nb_classes, activation=output_activation))

    return model
```

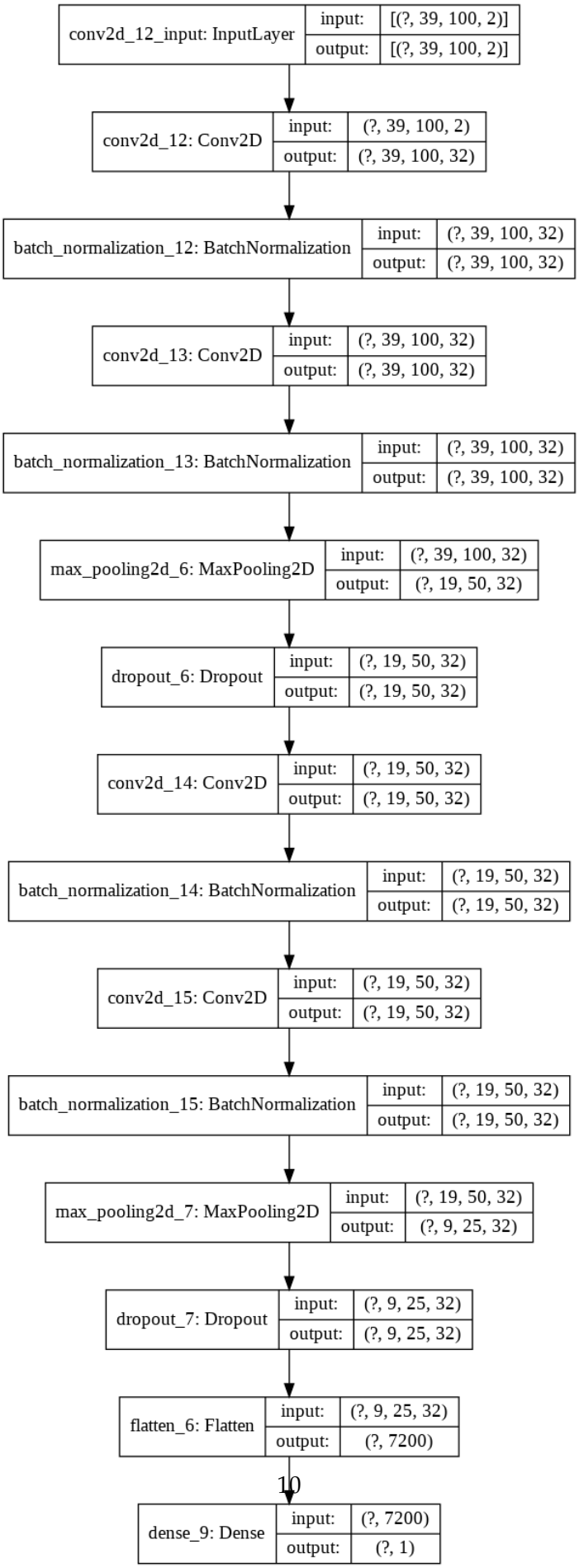
[183]:
```
model = create_cnn(input_shape=X_train.shape[1:], nb_classes=1, nb_blocks=2,
                   nb_filters=32, weight_decay=1e-4, padding='same', dropout=.
↪3, output_activation='sigmoid')
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 39, 100, 32)       608
_________________________________________________________________
batch_normalization_12 (Batc (None, 39, 100, 32)       128
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 39, 100, 32)       9248
_________________________________________________________________
batch_normalization_13 (Batc (None, 39, 100, 32)       128
```

```
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 19, 50, 32)        0
_________________________________________________________________
dropout_6 (Dropout)          (None, 19, 50, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 19, 50, 32)        9248
_________________________________________________________________
batch_normalization_14 (Batc (None, 19, 50, 32)        128
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 19, 50, 32)        9248
_________________________________________________________________
batch_normalization_15 (Batc (None, 19, 50, 32)        128
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 9, 25, 32)         0
_________________________________________________________________
dropout_7 (Dropout)          (None, 9, 25, 32)         0
_________________________________________________________________
flatten_6 (Flatten)          (None, 7200)              0
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 7201
=================================================================
Total params: 36,065
Trainable params: 35,809
Non-trainable params: 256
_________________________________________________________________
```

[184]:
```
# plot the graph  of the model and save to file
tf.keras.utils.plot_model(
    model,
    to_file='model.png', show_shapes=True, show_layer_names=True)
```

[184]:


conv2d_12_input: InputLayer
input: [(?, 39, 100, 2)]
output: [(?, 39, 100, 2)]
conv2d_12: Conv2D
input: (?, 39, 100, 2)
output: (?, 39, 100, 32)
batch_normalization_12: BatchNormalization
input: (?, 39, 100, 32)
output: (?, 39, 100, 32)
conv2d_13: Conv2D
input: (?, 39, 100, 32)
output: (?, 39, 100, 32)
batch_normalization_13: BatchNormalization
input: (?, 39, 100, 32)
output: (?, 39, 100, 32)
max_pooling2d_6: MaxPooling2D
input: (?, 39, 100, 32)
output: (?, 19, 50, 32)
dropout_6: Dropout
input: (?, 19, 50, 32)
output: (?, 19, 50, 32)
conv2d_14: Conv2D
input: (?, 19, 50, 32)
output: (?, 19, 50, 32)
batch_normalization_14: BatchNormalization
input: (?, 19, 50, 32)
output: (?, 19, 50, 32)
conv2d_15: Conv2D
input: (?, 19, 50, 32)
output: (?, 19, 50, 32)
batch_normalization_15: BatchNormalization
input: (?, 19, 50, 32)
output: (?, 19, 50, 32)
max_pooling2d_7: MaxPooling2D
input: (?, 19, 50, 32)
output: (?, 9, 25, 32)
dropout_7: Dropout
input: (?, 9, 25, 32)
output: (?, 9, 25, 32)
flatten_6: Flatten
input: (?, 9, 25, 32)
output: (?, 7200)
dense_9: Dense
input: (?, 7200)
output: (?, 1)

[185]:
```
def plot_history(history):

    import pandas as pd
    import matplotlib.pyplot as plt
    import tensorflow

    #------------------------------------------------------------
    # Retrieve results on training and validation data sets
    # for each training epoch
    #------------------------------------------------------------

    # check the current version
    if int(tensorflow.__version__.split('.')[0]) > 1:
        acc_key = 'accuracy'
    else:
        acc_key = 'acc'

    acc      = history.history[acc_key]
    val_acc  = history.history['val_' + acc_key]
    loss     = history.history['loss']
    val_loss = history.history['val_loss']
    epochs   = range(1,len(acc)+1)

    plt.rcParams['font.size'] = 16
    plt.rcParams['axes.spines.right'] = False
    plt.rcParams['axes.spines.top'] = False

    fig, (ax1, ax2) = plt.subplots(1,2,figsize=(10,5))

    #------------------------------------------------
    # Plot training and validation accuracy per epoch
    #------------------------------------------------
    ax1.plot(epochs, acc,  label='Training accuracy')
    ax1.plot(epochs, val_acc,  label='Validation accuracy')
    ax1.set_title('Accuracy')
    ax1.set_xlabel('epoch')
    ax1.set_ylabel('accuracy')
    ax1.set_ylim(0.5,1.2)

    #------------------------------------------------
    # Plot training and validation loss per epoch
    #------------------------------------------------

    ax2.plot(epochs, loss,  label='Training Loss')
    ax2.plot(epochs, val_loss,  label='Validation Loss')
```

```
    ax2.set_title('Loss')
    ax2.set_xlabel('epoch')
    ax2.set_ylabel('loss')
    ax2.set_ylim(-0.3,3)
    ax2.legend()

    fig.tight_layout()
    plt.show()
```

[186]:
```
opt_rms = tf.keras.optimizers.RMSprop(lr=0.01, decay=1e-6)
model.compile(loss='binary_crossentropy', optimizer='RMSprop',␣
 ↪metrics=['accuracy'])
batch_size = 64
epochs = 75
```

[187]:
```
%%time

history = model.fit(X_train, y_train, batch_size=batch_size,
                                    epochs=epochs,
                                    verbose=1,
                                    validation_data=(X_val, y_val))
```

```
Epoch 1/75
31/31 [==============================] - 1s 25ms/step - loss: 0.2546 - accuracy:
0.9215 - val_loss: 0.8524 - val_accuracy: 0.4221
Epoch 2/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0396 - accuracy:
0.9897 - val_loss: 2.0774 - val_accuracy: 0.4221
Epoch 3/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0125 - accuracy:
1.0000 - val_loss: 4.0662 - val_accuracy: 0.4221
Epoch 4/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0164 - accuracy:
0.9969 - val_loss: 7.4341 - val_accuracy: 0.4221
Epoch 5/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0100 - accuracy:
1.0000 - val_loss: 9.0041 - val_accuracy: 0.4221
Epoch 6/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0106 - accuracy:
0.9995 - val_loss: 11.0709 - val_accuracy: 0.4221
Epoch 7/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0094 - accuracy:
1.0000 - val_loss: 10.7740 - val_accuracy: 0.4221
Epoch 8/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0085 - accuracy:
1.0000 - val_loss: 12.2204 - val_accuracy: 0.4221
Epoch 9/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0071 - accuracy:
```

```
1.0000 - val_loss: 11.3373 - val_accuracy: 0.4221
Epoch 10/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0062 - accuracy:
1.0000 - val_loss: 15.7293 - val_accuracy: 0.4221
Epoch 11/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0051 - accuracy:
1.0000 - val_loss: 15.1843 - val_accuracy: 0.4221
Epoch 12/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0050 - accuracy:
0.9995 - val_loss: 16.5557 - val_accuracy: 0.4221
Epoch 13/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0028 - accuracy:
1.0000 - val_loss: 16.8077 - val_accuracy: 0.4221
Epoch 14/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0027 - accuracy:
1.0000 - val_loss: 16.2781 - val_accuracy: 0.4221
Epoch 15/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0023 - accuracy:
1.0000 - val_loss: 14.7869 - val_accuracy: 0.4221
Epoch 16/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0055 - accuracy:
0.9995 - val_loss: 17.4724 - val_accuracy: 0.4221
Epoch 17/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0017 - accuracy:
1.0000 - val_loss: 14.6851 - val_accuracy: 0.4245
Epoch 18/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0017 - accuracy:
1.0000 - val_loss: 13.9573 - val_accuracy: 0.4269
Epoch 19/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0014 - accuracy:
1.0000 - val_loss: 10.6605 - val_accuracy: 0.4221
Epoch 20/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0038 - accuracy:
0.9990 - val_loss: 8.5417 - val_accuracy: 0.4269
Epoch 21/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0085 - accuracy:
0.9979 - val_loss: 8.6719 - val_accuracy: 0.4508
Epoch 22/75
31/31 [==============================] - 1s 18ms/step - loss: 7.5220e-04 -
accuracy: 1.0000 - val_loss: 4.1706 - val_accuracy: 0.5564
Epoch 23/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0062 - accuracy:
0.9985 - val_loss: 0.1682 - val_accuracy: 0.9448
Epoch 24/75
31/31 [==============================] - 1s 18ms/step - loss: 7.9326e-04 -
accuracy: 1.0000 - val_loss: 0.0469 - val_accuracy: 0.9880
Epoch 25/75
31/31 [==============================] - 1s 18ms/step - loss: 7.4797e-04 -
```

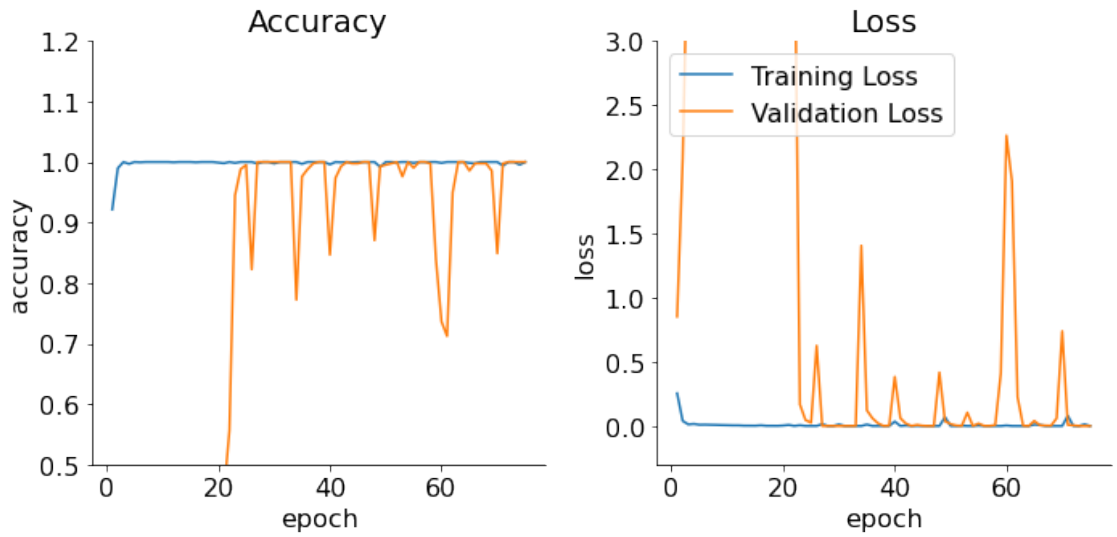
```
accuracy: 1.0000 - val_loss: 0.0250 - val_accuracy: 0.9952
Epoch 26/75
31/31 [==============================] - 1s 18ms/step - loss: 6.4682e-04 -
accuracy: 1.0000 - val_loss: 0.6259 - val_accuracy: 0.8225
Epoch 27/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0150 - accuracy:
0.9974 - val_loss: 5.3658e-04 - val_accuracy: 1.0000
Epoch 28/75
31/31 [==============================] - 1s 18ms/step - loss: 5.0857e-04 -
accuracy: 1.0000 - val_loss: 5.0180e-04 - val_accuracy: 1.0000
Epoch 29/75
31/31 [==============================] - 1s 18ms/step - loss: 4.8441e-04 -
accuracy: 1.0000 - val_loss: 4.7739e-04 - val_accuracy: 1.0000
Epoch 30/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0126 - accuracy:
0.9974 - val_loss: 5.0315e-04 - val_accuracy: 1.0000
Epoch 31/75
31/31 [==============================] - 1s 18ms/step - loss: 5.5060e-04 -
accuracy: 1.0000 - val_loss: 5.0042e-04 - val_accuracy: 1.0000
Epoch 32/75
31/31 [==============================] - 1s 18ms/step - loss: 4.9370e-04 -
accuracy: 1.0000 - val_loss: 4.7644e-04 - val_accuracy: 1.0000
Epoch 33/75
31/31 [==============================] - 1s 18ms/step - loss: 4.4598e-04 -
accuracy: 1.0000 - val_loss: 9.2165e-04 - val_accuracy: 1.0000
Epoch 34/75
31/31 [==============================] - 1s 18ms/step - loss: 3.2269e-04 -
accuracy: 1.0000 - val_loss: 1.4055 - val_accuracy: 0.7722
Epoch 35/75
31/31 [==============================] - 1s 19ms/step - loss: 0.0120 - accuracy:
0.9969 - val_loss: 0.1224 - val_accuracy: 0.9760
Epoch 36/75
31/31 [==============================] - 1s 18ms/step - loss: 4.0847e-04 -
accuracy: 1.0000 - val_loss: 0.0604 - val_accuracy: 0.9880
Epoch 37/75
31/31 [==============================] - 1s 18ms/step - loss: 3.5156e-04 -
accuracy: 1.0000 - val_loss: 0.0213 - val_accuracy: 0.9976
Epoch 38/75
31/31 [==============================] - 1s 18ms/step - loss: 3.1706e-04 -
accuracy: 1.0000 - val_loss: 3.8790e-04 - val_accuracy: 1.0000
Epoch 39/75
31/31 [==============================] - 1s 18ms/step - loss: 2.5976e-04 -
accuracy: 1.0000 - val_loss: 0.0033 - val_accuracy: 1.0000
Epoch 40/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0350 - accuracy:
0.9959 - val_loss: 0.3838 - val_accuracy: 0.8465
Epoch 41/75
31/31 [==============================] - 1s 18ms/step - loss: 6.4369e-04 -
```

```
accuracy: 1.0000 - val_loss: 0.0613 - val_accuracy: 0.9736
Epoch 42/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0037 - accuracy:
0.9995 - val_loss: 0.0188 - val_accuracy: 0.9928
Epoch 43/75
31/31 [==============================] - 1s 18ms/step - loss: 1.9870e-04 -
accuracy: 1.0000 - val_loss: 8.0512e-04 - val_accuracy: 1.0000
Epoch 44/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0014 - accuracy:
0.9995 - val_loss: 0.0089 - val_accuracy: 0.9976
Epoch 45/75
31/31 [==============================] - 1s 18ms/step - loss: 2.5271e-04 -
accuracy: 1.0000 - val_loss: 0.0024 - val_accuracy: 0.9976
Epoch 46/75
31/31 [==============================] - 1s 18ms/step - loss: 3.5638e-04 -
accuracy: 1.0000 - val_loss: 2.3836e-04 - val_accuracy: 1.0000
Epoch 47/75
31/31 [==============================] - 1s 18ms/step - loss: 2.2246e-04 -
accuracy: 1.0000 - val_loss: 7.8792e-04 - val_accuracy: 1.0000
Epoch 48/75
31/31 [==============================] - 1s 18ms/step - loss: 1.5733e-04 -
accuracy: 1.0000 - val_loss: 0.4175 - val_accuracy: 0.8705
Epoch 49/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0690 - accuracy:
0.9918 - val_loss: 0.0374 - val_accuracy: 0.9928
Epoch 50/75
31/31 [==============================] - 1s 18ms/step - loss: 1.4807e-04 -
accuracy: 1.0000 - val_loss: 0.0152 - val_accuracy: 0.9952
Epoch 51/75
31/31 [==============================] - 1s 18ms/step - loss: 1.3734e-04 -
accuracy: 1.0000 - val_loss: 0.0042 - val_accuracy: 0.9976
Epoch 52/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0027 - accuracy:
0.9995 - val_loss: 5.2941e-04 - val_accuracy: 1.0000
Epoch 53/75
31/31 [==============================] - 1s 18ms/step - loss: 2.4216e-04 -
accuracy: 1.0000 - val_loss: 0.1046 - val_accuracy: 0.9760
Epoch 54/75
31/31 [==============================] - 1s 18ms/step - loss: 2.0108e-04 -
accuracy: 1.0000 - val_loss: 1.9062e-04 - val_accuracy: 1.0000
Epoch 55/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0038 - accuracy:
0.9985 - val_loss: 0.0199 - val_accuracy: 0.9904
Epoch 56/75
31/31 [==============================] - 1s 18ms/step - loss: 3.3085e-04 -
accuracy: 1.0000 - val_loss: 0.0014 - val_accuracy: 1.0000
Epoch 57/75
31/31 [==============================] - 1s 18ms/step - loss: 1.8719e-04 -
```

```
accuracy: 1.0000 - val_loss: 5.2132e-04 - val_accuracy: 1.0000
Epoch 58/75
31/31 [==============================] - 1s 18ms/step - loss: 1.7264e-04 -
accuracy: 1.0000 - val_loss: 0.0078 - val_accuracy: 0.9976
Epoch 59/75
31/31 [==============================] - 1s 18ms/step - loss: 1.2344e-04 -
accuracy: 1.0000 - val_loss: 0.4106 - val_accuracy: 0.8393
Epoch 60/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0043 - accuracy:
0.9985 - val_loss: 2.2640 - val_accuracy: 0.7362
Epoch 61/75
31/31 [==============================] - 1s 18ms/step - loss: 4.0554e-04 -
accuracy: 1.0000 - val_loss: 1.9065 - val_accuracy: 0.7122
Epoch 62/75
31/31 [==============================] - 1s 18ms/step - loss: 1.6539e-04 -
accuracy: 1.0000 - val_loss: 0.2259 - val_accuracy: 0.9496
Epoch 63/75
31/31 [==============================] - 1s 18ms/step - loss: 1.4166e-04 -
accuracy: 1.0000 - val_loss: 0.0020 - val_accuracy: 1.0000
Epoch 64/75
31/31 [==============================] - 1s 18ms/step - loss: 1.1692e-04 -
accuracy: 1.0000 - val_loss: 1.5534e-04 - val_accuracy: 1.0000
Epoch 65/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0114 - accuracy:
0.9985 - val_loss: 0.0398 - val_accuracy: 0.9856
Epoch 66/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0080 - accuracy:
0.9979 - val_loss: 0.0059 - val_accuracy: 0.9976
Epoch 67/75
31/31 [==============================] - 1s 18ms/step - loss: 1.3872e-04 -
accuracy: 1.0000 - val_loss: 0.0036 - val_accuracy: 0.9976
Epoch 68/75
31/31 [==============================] - 1s 18ms/step - loss: 1.3246e-04 -
accuracy: 1.0000 - val_loss: 0.0024 - val_accuracy: 0.9976
Epoch 69/75
31/31 [==============================] - 1s 18ms/step - loss: 1.2614e-04 -
accuracy: 1.0000 - val_loss: 0.0603 - val_accuracy: 0.9856
Epoch 70/75
31/31 [==============================] - 1s 18ms/step - loss: 8.4967e-05 -
accuracy: 1.0000 - val_loss: 0.7390 - val_accuracy: 0.8489
Epoch 71/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0794 - accuracy:
0.9938 - val_loss: 0.0092 - val_accuracy: 0.9976
Epoch 72/75
31/31 [==============================] - 1s 18ms/step - loss: 3.4712e-04 -
accuracy: 1.0000 - val_loss: 0.0050 - val_accuracy: 1.0000
Epoch 73/75
31/31 [==============================] - 1s 19ms/step - loss: 9.9605e-05 -
```

```
accuracy: 1.0000 - val_loss: 3.8950e-04 - val_accuracy: 1.0000
Epoch 74/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0132 - accuracy:
0.9954 - val_loss: 1.4575e-04 - val_accuracy: 1.0000
Epoch 75/75
31/31 [==============================] - 1s 18ms/step - loss: 1.7373e-04 -
accuracy: 1.0000 - val_loss: 1.4748e-04 - val_accuracy: 1.0000
CPU times: user 31.7 s, sys: 8.7 s, total: 40.4 s
Wall time: 44.5 s
```

[188]:
```
plot_history(history)
```



[189]:
```
test_loss, test_accuracy = model.evaluate(X_test, y_test, batch_size=128,␣
↪verbose=2)
print('Accuracy on test dataset:', test_accuracy)
```

```
4/4 - 0s - loss: 1.4495e-04 - accuracy: 1.0000
Accuracy on test dataset: 1.0
```

After 75 epochs the validation curves have not stabilized stabilized. Let's try a smaller learning rate.

2. Same CNN CIFAR-10 style model, but 10x smaller learning rate

[190]:
```
opt_rms = tf.keras.optimizers.RMSprop(lr=0.001, decay=1e-6)
model.compile(loss='binary_crossentropy', optimizer='RMSprop',␣
↪metrics=['accuracy'])
batch_size = 64
epochs = 75
```

[191]:
```
%%time

history = model.fit(X_train, y_train, batch_size=batch_size,
                                    epochs=epochs,
                                    verbose=1,
                                    validation_data=(X_val, y_val))
```

```
Epoch 1/75
31/31 [==============================] - 1s 23ms/step - loss: 0.0210 - accuracy:
0.9959 - val_loss: 0.0665 - val_accuracy: 0.9640
Epoch 2/75
31/31 [==============================] - 1s 18ms/step - loss: 5.6628e-04 -
accuracy: 1.0000 - val_loss: 8.7125e-04 - val_accuracy: 1.0000
Epoch 3/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0032 - accuracy:
0.9985 - val_loss: 5.1240 - val_accuracy: 0.6307
Epoch 4/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0012 - accuracy:
0.9995 - val_loss: 2.5072 - val_accuracy: 0.7242
Epoch 5/75
31/31 [==============================] - 1s 18ms/step - loss: 2.2802e-04 -
accuracy: 1.0000 - val_loss: 0.7324 - val_accuracy: 0.8825
Epoch 6/75
31/31 [==============================] - 1s 18ms/step - loss: 1.2605e-04 -
accuracy: 1.0000 - val_loss: 0.0071 - val_accuracy: 0.9952
Epoch 7/75
31/31 [==============================] - 1s 18ms/step - loss: 1.0173e-04 -
accuracy: 1.0000 - val_loss: 9.2721e-05 - val_accuracy: 1.0000
Epoch 8/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0014 - accuracy:
0.9990 - val_loss: 4.9557 - val_accuracy: 0.5803
Epoch 9/75
31/31 [==============================] - 1s 19ms/step - loss: 0.0050 - accuracy:
0.9985 - val_loss: 2.1696e-04 - val_accuracy: 1.0000
Epoch 10/75
31/31 [==============================] - 1s 18ms/step - loss: 9.0866e-05 -
accuracy: 1.0000 - val_loss: 0.0010 - val_accuracy: 1.0000
Epoch 11/75
31/31 [==============================] - 1s 19ms/step - loss: 6.9378e-04 -
accuracy: 1.0000 - val_loss: 0.0010 - val_accuracy: 1.0000
Epoch 12/75
31/31 [==============================] - 1s 19ms/step - loss: 1.0389e-04 -
accuracy: 1.0000 - val_loss: 3.4357e-04 - val_accuracy: 1.0000
Epoch 13/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0083 - accuracy:
0.9990 - val_loss: 0.0862 - val_accuracy: 0.9856
Epoch 14/75
```

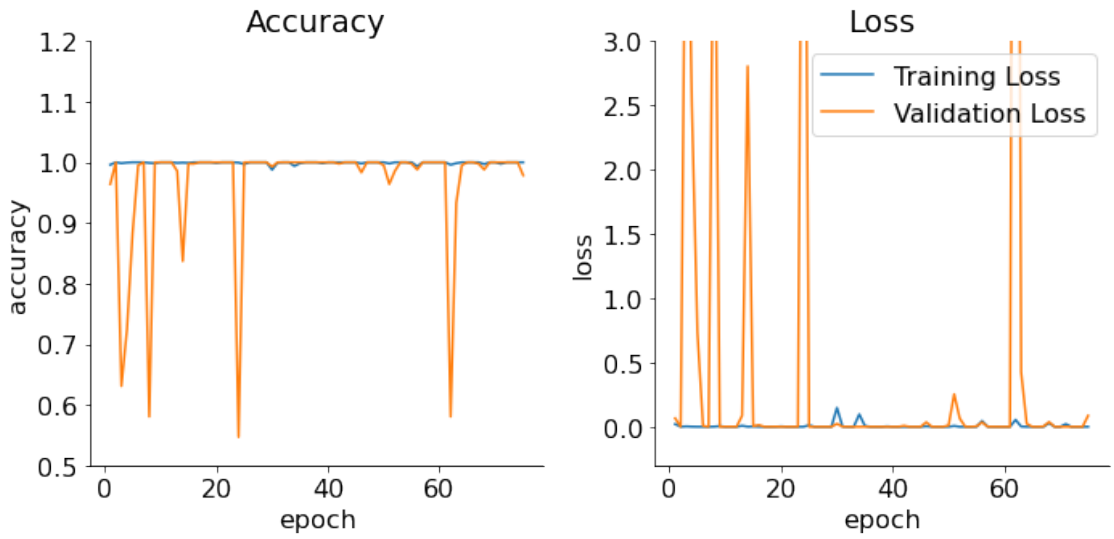
```
31/31 [==============================] - 1s 19ms/step - loss: 9.3136e-04 -
accuracy: 0.9995 - val_loss: 2.8046 - val_accuracy: 0.8369
Epoch 15/75
31/31 [==============================] - 1s 19ms/step - loss: 0.0024 - accuracy:
0.9990 - val_loss: 0.0052 - val_accuracy: 0.9976
Epoch 16/75
31/31 [==============================] - 1s 18ms/step - loss: 2.0616e-04 -
accuracy: 1.0000 - val_loss: 0.0137 - val_accuracy: 0.9976
Epoch 17/75
31/31 [==============================] - 1s 18ms/step - loss: 3.4519e-04 -
accuracy: 1.0000 - val_loss: 2.9107e-04 - val_accuracy: 1.0000
Epoch 18/75
31/31 [==============================] - 1s 18ms/step - loss: 1.6873e-04 -
accuracy: 1.0000 - val_loss: 1.5118e-04 - val_accuracy: 1.0000
Epoch 19/75
31/31 [==============================] - 1s 18ms/step - loss: 1.3695e-04 -
accuracy: 1.0000 - val_loss: 0.0012 - val_accuracy: 1.0000
Epoch 20/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0028 - accuracy:
0.9990 - val_loss: 1.8545e-04 - val_accuracy: 1.0000
Epoch 21/75
31/31 [==============================] - 1s 18ms/step - loss: 2.1028e-04 -
accuracy: 1.0000 - val_loss: 1.7800e-04 - val_accuracy: 1.0000
Epoch 22/75
31/31 [==============================] - 1s 18ms/step - loss: 1.6841e-04 -
accuracy: 1.0000 - val_loss: 1.5327e-04 - val_accuracy: 1.0000
Epoch 23/75
31/31 [==============================] - 1s 19ms/step - loss: 1.2766e-04 -
accuracy: 1.0000 - val_loss: 4.4233e-04 - val_accuracy: 1.0000
Epoch 24/75
31/31 [==============================] - 1s 19ms/step - loss: 5.1215e-04 -
accuracy: 0.9995 - val_loss: 7.1062 - val_accuracy: 0.5468
Epoch 25/75
31/31 [==============================] - 1s 19ms/step - loss: 0.0121 - accuracy:
0.9969 - val_loss: 1.0798e-04 - val_accuracy: 1.0000
Epoch 26/75
31/31 [==============================] - 1s 19ms/step - loss: 9.6942e-05 -
accuracy: 1.0000 - val_loss: 9.6254e-05 - val_accuracy: 1.0000
Epoch 27/75
31/31 [==============================] - 1s 18ms/step - loss: 9.6927e-05 -
accuracy: 1.0000 - val_loss: 8.7441e-05 - val_accuracy: 1.0000
Epoch 28/75
31/31 [==============================] - 1s 18ms/step - loss: 8.4528e-05 -
accuracy: 1.0000 - val_loss: 7.6252e-05 - val_accuracy: 1.0000
Epoch 29/75
31/31 [==============================] - 1s 18ms/step - loss: 6.3443e-05 -
accuracy: 1.0000 - val_loss: 6.0390e-05 - val_accuracy: 1.0000
Epoch 30/75
```

```
31/31 [==============================] - 1s 18ms/step - loss: 0.1475 - accuracy:
0.9877 - val_loss: 0.0235 - val_accuracy: 0.9928
Epoch 31/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0015 - accuracy:
0.9995 - val_loss: 0.0014 - val_accuracy: 1.0000
Epoch 32/75
31/31 [==============================] - 1s 18ms/step - loss: 5.6837e-05 -
accuracy: 1.0000 - val_loss: 1.3315e-04 - val_accuracy: 1.0000
Epoch 33/75
31/31 [==============================] - 1s 19ms/step - loss: 5.5831e-05 -
accuracy: 1.0000 - val_loss: 6.4606e-05 - val_accuracy: 1.0000
Epoch 34/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0974 - accuracy:
0.9938 - val_loss: 9.3720e-05 - val_accuracy: 1.0000
Epoch 35/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0068 - accuracy:
0.9985 - val_loss: 9.9964e-04 - val_accuracy: 1.0000
Epoch 36/75
31/31 [==============================] - 1s 18ms/step - loss: 1.0694e-04 -
accuracy: 1.0000 - val_loss: 3.3615e-04 - val_accuracy: 1.0000
Epoch 37/75
31/31 [==============================] - 1s 18ms/step - loss: 1.0001e-04 -
accuracy: 1.0000 - val_loss: 1.5204e-04 - val_accuracy: 1.0000
Epoch 38/75
31/31 [==============================] - 1s 18ms/step - loss: 8.7658e-05 -
accuracy: 1.0000 - val_loss: 9.7969e-05 - val_accuracy: 1.0000
Epoch 39/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0030 - accuracy:
0.9985 - val_loss: 1.1867e-04 - val_accuracy: 1.0000
Epoch 40/75
31/31 [==============================] - 1s 18ms/step - loss: 1.1626e-04 -
accuracy: 1.0000 - val_loss: 1.1273e-04 - val_accuracy: 1.0000
Epoch 41/75
31/31 [==============================] - 1s 18ms/step - loss: 1.0955e-04 -
accuracy: 1.0000 - val_loss: 1.0253e-04 - val_accuracy: 1.0000
Epoch 42/75
31/31 [==============================] - 1s 18ms/step - loss: 9.1743e-05 -
accuracy: 1.0000 - val_loss: 0.0079 - val_accuracy: 0.9976
Epoch 43/75
31/31 [==============================] - 1s 18ms/step - loss: 1.9900e-04 -
accuracy: 1.0000 - val_loss: 1.1644e-04 - val_accuracy: 1.0000
Epoch 44/75
31/31 [==============================] - 1s 18ms/step - loss: 1.0137e-04 -
accuracy: 1.0000 - val_loss: 1.4662e-04 - val_accuracy: 1.0000
Epoch 45/75
31/31 [==============================] - 1s 18ms/step - loss: 6.2126e-05 -
accuracy: 1.0000 - val_loss: 3.0339e-04 - val_accuracy: 1.0000
Epoch 46/75
```

```
31/31 [==============================] - 1s 18ms/step - loss: 0.0043 - accuracy:
0.9979 - val_loss: 0.0355 - val_accuracy: 0.9832
Epoch 47/75
31/31 [==============================] - 1s 18ms/step - loss: 1.5216e-04 -
accuracy: 1.0000 - val_loss: 7.4646e-04 - val_accuracy: 1.0000
Epoch 48/75
31/31 [==============================] - 1s 18ms/step - loss: 7.7314e-05 -
accuracy: 1.0000 - val_loss: 9.9037e-05 - val_accuracy: 1.0000
Epoch 49/75
31/31 [==============================] - 1s 18ms/step - loss: 6.7257e-05 -
accuracy: 1.0000 - val_loss: 6.2177e-05 - val_accuracy: 1.0000
Epoch 50/75
31/31 [==============================] - 1s 18ms/step - loss: 5.5262e-05 -
accuracy: 1.0000 - val_loss: 0.0110 - val_accuracy: 0.9952
Epoch 51/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0069 - accuracy:
0.9979 - val_loss: 0.2541 - val_accuracy: 0.9640
Epoch 52/75
31/31 [==============================] - 1s 18ms/step - loss: 7.3561e-05 -
accuracy: 1.0000 - val_loss: 0.0651 - val_accuracy: 0.9856
Epoch 53/75
31/31 [==============================] - 1s 18ms/step - loss: 9.7545e-05 -
accuracy: 1.0000 - val_loss: 6.7233e-05 - val_accuracy: 1.0000
Epoch 54/75
31/31 [==============================] - 1s 18ms/step - loss: 6.1313e-05 -
accuracy: 1.0000 - val_loss: 5.6364e-05 - val_accuracy: 1.0000
Epoch 55/75
31/31 [==============================] - 1s 19ms/step - loss: 4.6384e-05 -
accuracy: 1.0000 - val_loss: 0.0037 - val_accuracy: 0.9976
Epoch 56/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0453 - accuracy:
0.9933 - val_loss: 0.0353 - val_accuracy: 0.9880
Epoch 57/75
31/31 [==============================] - 1s 18ms/step - loss: 7.4547e-04 -
accuracy: 1.0000 - val_loss: 9.3488e-04 - val_accuracy: 1.0000
Epoch 58/75
31/31 [==============================] - 1s 18ms/step - loss: 7.3408e-05 -
accuracy: 1.0000 - val_loss: 3.2313e-04 - val_accuracy: 1.0000
Epoch 59/75
31/31 [==============================] - 1s 18ms/step - loss: 7.5535e-05 -
accuracy: 1.0000 - val_loss: 6.1638e-05 - val_accuracy: 1.0000
Epoch 60/75
31/31 [==============================] - 1s 19ms/step - loss: 5.7697e-05 -
accuracy: 1.0000 - val_loss: 5.2632e-05 - val_accuracy: 1.0000
Epoch 61/75
31/31 [==============================] - 1s 18ms/step - loss: 4.4204e-05 -
accuracy: 1.0000 - val_loss: 4.0947e-05 - val_accuracy: 1.0000
Epoch 62/75
```

```
31/31 [==============================] - 1s 18ms/step - loss: 0.0547 - accuracy:
0.9959 - val_loss: 12.1918 - val_accuracy: 0.5803
Epoch 63/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0024 - accuracy:
0.9985 - val_loss: 0.4257 - val_accuracy: 0.9329
Epoch 64/75
31/31 [==============================] - 1s 18ms/step - loss: 9.6257e-05 -
accuracy: 1.0000 - val_loss: 0.0236 - val_accuracy: 0.9952
Epoch 65/75
31/31 [==============================] - 1s 18ms/step - loss: 8.5153e-05 -
accuracy: 1.0000 - val_loss: 3.6481e-04 - val_accuracy: 1.0000
Epoch 66/75
31/31 [==============================] - 1s 18ms/step - loss: 7.3424e-05 -
accuracy: 1.0000 - val_loss: 6.3134e-05 - val_accuracy: 1.0000
Epoch 67/75
31/31 [==============================] - 1s 18ms/step - loss: 4.8500e-05 -
accuracy: 1.0000 - val_loss: 0.0030 - val_accuracy: 0.9976
Epoch 68/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0286 - accuracy:
0.9969 - val_loss: 0.0377 - val_accuracy: 0.9880
Epoch 69/75
31/31 [==============================] - 1s 18ms/step - loss: 9.7575e-05 -
accuracy: 1.0000 - val_loss: 8.8654e-05 - val_accuracy: 1.0000
Epoch 70/75
31/31 [==============================] - 1s 18ms/step - loss: 6.4968e-05 -
accuracy: 1.0000 - val_loss: 1.4772e-04 - val_accuracy: 1.0000
Epoch 71/75
31/31 [==============================] - 1s 18ms/step - loss: 0.0215 - accuracy:
0.9974 - val_loss: 9.4186e-04 - val_accuracy: 1.0000
Epoch 72/75
31/31 [==============================] - 1s 18ms/step - loss: 9.6679e-05 -
accuracy: 1.0000 - val_loss: 1.0227e-04 - val_accuracy: 1.0000
Epoch 73/75
31/31 [==============================] - 1s 18ms/step - loss: 9.1068e-05 -
accuracy: 1.0000 - val_loss: 8.7135e-05 - val_accuracy: 1.0000
Epoch 74/75
31/31 [==============================] - 1s 18ms/step - loss: 8.1063e-05 -
accuracy: 1.0000 - val_loss: 7.2624e-05 - val_accuracy: 1.0000
Epoch 75/75
31/31 [==============================] - 1s 18ms/step - loss: 5.1860e-04 -
accuracy: 1.0000 - val_loss: 0.0865 - val_accuracy: 0.9784
CPU times: user 31.7 s, sys: 9.1 s, total: 40.8 s
Wall time: 44.8 s
```

[192]:
```
plot_history(history)
```



[193]:
```
test_loss, test_accuracy = model.evaluate(X_test, y_test, batch_size=128,␣
↪verbose=2)
print('Accuracy on test dataset:', test_accuracy)
```

```
4/4 - 0s - loss: 0.2530 - accuracy: 0.9665
Accuracy on test dataset: 0.9665071964263916
```

While the training curves above are smooth, the extreme variability in the validation curves and the lack of a convergence gives cause for concern. Evidently, the number of epochs run will randomly detemine the performance on validation and test data. We need a better method.

3. Transfer earning: VGG-16 (fixed weights) plus a classifier head (trainable weights)

Since VGG-16 expects a three-channel input, we will place the average of the two channels in the third channel.

[194]:
```
print(X_test.shape)
X3_test = np.zeros((X_test.shape[0], X_test.shape[1], X_test.shape[2], X_test.
↪shape[3]+1))
X3_test[..., 0] = X_test[..., 0]
X3_test[..., 1] = X_test[..., 1]
X3_test[..., 2] = (X_test[..., 1] + X_test[..., 1]) / 2.0
print(X3_test.shape)
```

```
(418, 39, 100, 2)
(418, 39, 100, 3)
```

[195]:
```
print(X_val.shape)
X3_val = np.zeros((X_val.shape[0], X_val.shape[1], X_val.shape[2], X_val.
↪shape[3]+1))
X3_val[..., 0] = X_val[..., 0]
X3_val[..., 1] = X_val[..., 1]
```

```
X3_val[..., 2] = (X_val[..., 1] + X_val[..., 1]) / 2.0
print(X3_val.shape)
```

```
(417, 39, 100, 2)
(417, 39, 100, 3)
```

[196]:
```
print(X_train.shape)
X3_train = np.zeros((X_train.shape[0], X_train.shape[1], X_train.shape[2],␣
 ↪X_train.shape[3]+1))
X3_train[..., 0] = X_train[..., 0]
X3_train[..., 1] = X_train[..., 1]
X3_train[..., 2] = (X_train[..., 1] + X_train[..., 1]) / 2.0
print(X3_train.shape)
```

```
(1949, 39, 100, 2)
(1949, 39, 100, 3)
```

Further preprocessing found to not be necessary. We thought about trying to preprocess the data in the exact way expected by VGG-16 (subtracting average value from each pixel) and rescaling the image from 39 x 100 to 228 x 228 but this is expected to have minimal effect. We see below that exceptional performance is already obtained without those additional steps.

[197]:
```
# load pre-trained model from keras
from tensorflow.keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(X3_train.shape[1], X3_train.shape[2], 3))
```

[198]:
```
# print summary of convolutional base
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 39, 100, 3)]      0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 39, 100, 64)       1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 39, 100, 64)       36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 19, 50, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 19, 50, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 19, 50, 128)       147584
_________________________________________________________________
```

```
block2_pool (MaxPooling2D)   (None, 9, 25, 128)        0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 9, 25, 256)        295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 9, 25, 256)        590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 9, 25, 256)        590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 12, 256)        0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 4, 12, 512)        1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 4, 12, 512)        2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 4, 12, 512)        2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 2, 6, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 2, 6, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 2, 6, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 2, 6, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 3, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

[199]:
```
conv_base.trainable = False
```

[200]:
```
model = tf.keras.models.Sequential()

# add convolutional base as layer
model.add(conv_base)

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dense(1,activation="sigmoid"))

# display model summary
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
vgg16 (Functional)           (None, 1, 3, 512)         14714688
_________________________________________________________________
flatten_7 (Flatten)          (None, 1536)              0
_________________________________________________________________
dense_10 (Dense)             (None, 256)               393472
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 257
=================================================================
Total params: 15,108,417
Trainable params: 393,729
Non-trainable params: 14,714,688
_________________________________________________________________
```

[201]:
```
opt_rms = tf.keras.optimizers.RMSprop(lr=2e-5, decay=1e-6) # this is much␣
 ↪smaller lr than previous lr=0.01
model.compile(loss='binary_crossentropy', optimizer='RMSprop',␣
 ↪metrics=['accuracy'])
batch_size = 64
epochs = 75
```

[202]:
```
%%time

history = model.fit(X3_train, y_train, batch_size=batch_size,
                                    epochs=epochs,
                                    verbose=1,
                                    validation_data=(X3_val, y_val))
```

```
Epoch 1/75
31/31 [==============================] - 1s 35ms/step - loss: 0.3992 - accuracy:
0.8250 - val_loss: 0.1879 - val_accuracy: 0.9376
Epoch 2/75
31/31 [==============================] - 1s 30ms/step - loss: 0.2238 - accuracy:
0.9015 - val_loss: 0.1678 - val_accuracy: 0.9353
Epoch 3/75
31/31 [==============================] - 1s 30ms/step - loss: 0.1651 - accuracy:
0.9359 - val_loss: 0.1282 - val_accuracy: 0.9544
Epoch 4/75
31/31 [==============================] - 1s 30ms/step - loss: 0.1478 - accuracy:
0.9456 - val_loss: 0.2664 - val_accuracy: 0.8921
Epoch 5/75
31/31 [==============================] - 1s 30ms/step - loss: 0.1371 - accuracy:
0.9487 - val_loss: 0.0882 - val_accuracy: 0.9784
Epoch 6/75
31/31 [==============================] - 1s 30ms/step - loss: 0.1037 - accuracy:
0.9615 - val_loss: 0.0663 - val_accuracy: 0.9808
Epoch 7/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0914 - accuracy:
```

```
0.9702 - val_loss: 0.0908 - val_accuracy: 0.9568
Epoch 8/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0850 - accuracy:
0.9661 - val_loss: 0.0685 - val_accuracy: 0.9736
Epoch 9/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0671 - accuracy:
0.9774 - val_loss: 0.1337 - val_accuracy: 0.9353
Epoch 10/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0742 - accuracy:
0.9687 - val_loss: 0.0354 - val_accuracy: 0.9976
Epoch 11/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0614 - accuracy:
0.9774 - val_loss: 0.1702 - val_accuracy: 0.9281
Epoch 12/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0505 - accuracy:
0.9826 - val_loss: 0.0296 - val_accuracy: 0.9928
Epoch 13/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0529 - accuracy:
0.9846 - val_loss: 0.0244 - val_accuracy: 0.9952
Epoch 14/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0534 - accuracy:
0.9820 - val_loss: 0.0214 - val_accuracy: 0.9976
Epoch 15/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0507 - accuracy:
0.9831 - val_loss: 0.0193 - val_accuracy: 0.9976
Epoch 16/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0292 - accuracy:
0.9928 - val_loss: 0.0487 - val_accuracy: 0.9808
Epoch 17/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0319 - accuracy:
0.9882 - val_loss: 0.0303 - val_accuracy: 0.9952
Epoch 18/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0266 - accuracy:
0.9918 - val_loss: 0.0826 - val_accuracy: 0.9616
Epoch 19/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0253 - accuracy:
0.9928 - val_loss: 0.0140 - val_accuracy: 0.9976
Epoch 20/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0309 - accuracy:
0.9908 - val_loss: 0.2517 - val_accuracy: 0.8801
Epoch 21/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0202 - accuracy:
0.9949 - val_loss: 0.0816 - val_accuracy: 0.9640
Epoch 22/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0273 - accuracy:
0.9918 - val_loss: 0.0561 - val_accuracy: 0.9760
Epoch 23/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0249 - accuracy:
```

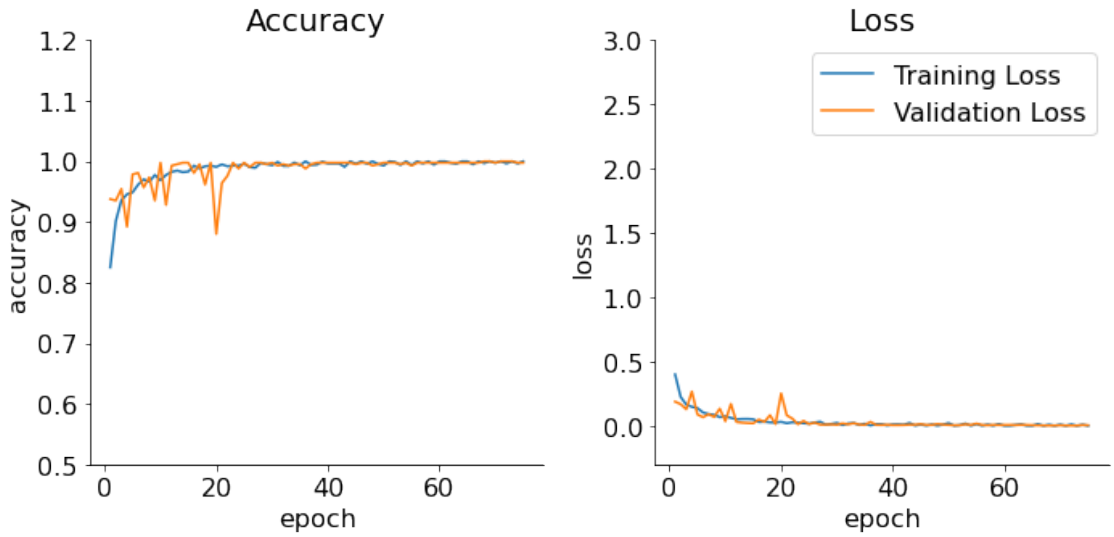
```
0.9928 - val_loss: 0.0120 - val_accuracy: 0.9976
Epoch 24/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0198 - accuracy:
0.9928 - val_loss: 0.0382 - val_accuracy: 0.9880
Epoch 25/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0168 - accuracy:
0.9959 - val_loss: 0.0132 - val_accuracy: 0.9976
Epoch 26/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0259 - accuracy:
0.9903 - val_loss: 0.0254 - val_accuracy: 0.9904
Epoch 27/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0318 - accuracy:
0.9892 - val_loss: 0.0086 - val_accuracy: 0.9976
Epoch 28/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0080 - accuracy:
0.9974 - val_loss: 0.0089 - val_accuracy: 0.9976
Epoch 29/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0146 - accuracy:
0.9954 - val_loss: 0.0107 - val_accuracy: 0.9952
Epoch 30/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0227 - accuracy:
0.9933 - val_loss: 0.0076 - val_accuracy: 0.9976
Epoch 31/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0056 - accuracy:
0.9990 - val_loss: 0.0187 - val_accuracy: 0.9928
Epoch 32/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0181 - accuracy:
0.9923 - val_loss: 0.0111 - val_accuracy: 0.9952
Epoch 33/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0216 - accuracy:
0.9923 - val_loss: 0.0201 - val_accuracy: 0.9928
Epoch 34/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0064 - accuracy:
0.9979 - val_loss: 0.0094 - val_accuracy: 0.9952
Epoch 35/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0163 - accuracy:
0.9933 - val_loss: 0.0067 - val_accuracy: 0.9952
Epoch 36/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0029 - accuracy:
1.0000 - val_loss: 0.0308 - val_accuracy: 0.9880
Epoch 37/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0127 - accuracy:
0.9938 - val_loss: 0.0069 - val_accuracy: 0.9952
Epoch 38/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0124 - accuracy:
0.9944 - val_loss: 0.0063 - val_accuracy: 0.9976
Epoch 39/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0026 - accuracy:
```

```
0.9995 - val_loss: 0.0081 - val_accuracy: 0.9976
Epoch 40/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0102 - accuracy:
0.9964 - val_loss: 0.0053 - val_accuracy: 0.9976
Epoch 41/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0109 - accuracy:
0.9964 - val_loss: 0.0053 - val_accuracy: 0.9976
Epoch 42/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0081 - accuracy:
0.9964 - val_loss: 0.0061 - val_accuracy: 0.9976
Epoch 43/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0223 - accuracy:
0.9908 - val_loss: 0.0087 - val_accuracy: 0.9976
Epoch 44/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0014 - accuracy:
1.0000 - val_loss: 0.0084 - val_accuracy: 0.9976
Epoch 45/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0101 - accuracy:
0.9959 - val_loss: 0.0129 - val_accuracy: 0.9952
Epoch 46/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0016 - accuracy:
1.0000 - val_loss: 0.0065 - val_accuracy: 0.9976
Epoch 47/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0103 - accuracy:
0.9959 - val_loss: 0.0067 - val_accuracy: 0.9976
Epoch 48/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0011 - accuracy:
1.0000 - val_loss: 0.0136 - val_accuracy: 0.9928
Epoch 49/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0092 - accuracy:
0.9954 - val_loss: 0.0092 - val_accuracy: 0.9952
Epoch 50/75
31/31 [==============================] - 1s 30ms/step - loss: 0.0214 - accuracy:
0.9928 - val_loss: 0.0066 - val_accuracy: 0.9976
Epoch 51/75
31/31 [==============================] - 1s 30ms/step - loss: 7.7169e-04 -
accuracy: 1.0000 - val_loss: 0.0043 - val_accuracy: 0.9976
Epoch 52/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0053 - accuracy:
0.9990 - val_loss: 0.0045 - val_accuracy: 0.9976
Epoch 53/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0176 - accuracy:
0.9938 - val_loss: 0.0119 - val_accuracy: 0.9952
Epoch 54/75
31/31 [==============================] - 1s 31ms/step - loss: 6.6445e-04 -
accuracy: 1.0000 - val_loss: 0.0062 - val_accuracy: 0.9976
Epoch 55/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0139 - accuracy:
```

```
0.9938 - val_loss: 0.0171 - val_accuracy: 0.9928
Epoch 56/75
31/31 [==============================] - 1s 31ms/step - loss: 8.4992e-04 -
accuracy: 1.0000 - val_loss: 0.0063 - val_accuracy: 0.9976
Epoch 57/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0098 - accuracy:
0.9964 - val_loss: 0.0045 - val_accuracy: 0.9976
Epoch 58/75
31/31 [==============================] - 1s 31ms/step - loss: 5.6181e-04 -
accuracy: 1.0000 - val_loss: 0.0067 - val_accuracy: 0.9976
Epoch 59/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0140 - accuracy:
0.9944 - val_loss: 0.0060 - val_accuracy: 0.9976
Epoch 60/75
31/31 [==============================] - 1s 31ms/step - loss: 4.4579e-04 -
accuracy: 1.0000 - val_loss: 0.0062 - val_accuracy: 0.9976
Epoch 61/75
31/31 [==============================] - 1s 31ms/step - loss: 5.3650e-04 -
accuracy: 1.0000 - val_loss: 0.0067 - val_accuracy: 0.9976
Epoch 62/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0066 - accuracy:
0.9974 - val_loss: 0.0057 - val_accuracy: 0.9976
Epoch 63/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0116 - accuracy:
0.9959 - val_loss: 0.0074 - val_accuracy: 0.9976
Epoch 64/75
31/31 [==============================] - 1s 31ms/step - loss: 6.3525e-04 -
accuracy: 1.0000 - val_loss: 0.0063 - val_accuracy: 0.9976
Epoch 65/75
31/31 [==============================] - 1s 30ms/step - loss: 4.6234e-04 -
accuracy: 1.0000 - val_loss: 0.0050 - val_accuracy: 0.9976
Epoch 66/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0137 - accuracy:
0.9949 - val_loss: 0.0037 - val_accuracy: 0.9976
Epoch 67/75
31/31 [==============================] - 1s 31ms/step - loss: 3.2343e-04 -
accuracy: 1.0000 - val_loss: 0.0028 - val_accuracy: 0.9976
Epoch 68/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0089 - accuracy:
0.9969 - val_loss: 0.0021 - val_accuracy: 1.0000
Epoch 69/75
31/31 [==============================] - 1s 30ms/step - loss: 2.4053e-04 -
accuracy: 1.0000 - val_loss: 0.0023 - val_accuracy: 1.0000
Epoch 70/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0095 - accuracy:
0.9974 - val_loss: 0.0040 - val_accuracy: 0.9976
Epoch 71/75
31/31 [==============================] - 1s 30ms/step - loss: 2.0591e-04 -
```

```
accuracy: 1.0000 - val_loss: 0.0025 - val_accuracy: 1.0000
Epoch 72/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0111 - accuracy:
0.9964 - val_loss: 0.0018 - val_accuracy: 1.0000
Epoch 73/75
31/31 [==============================] - 1s 31ms/step - loss: 2.4676e-04 -
accuracy: 1.0000 - val_loss: 0.0020 - val_accuracy: 1.0000
Epoch 74/75
31/31 [==============================] - 1s 31ms/step - loss: 0.0084 - accuracy:
0.9964 - val_loss: 0.0060 - val_accuracy: 0.9976
Epoch 75/75
31/31 [==============================] - 1s 31ms/step - loss: 2.0779e-04 -
accuracy: 1.0000 - val_loss: 0.0042 - val_accuracy: 0.9976
CPU times: user 22.9 s, sys: 3.34 s, total: 26.2 s
Wall time: 1min 13s
```

[203]:
```
plot_history(history)
```



[204]:
```
test_loss, test_accuracy = model.evaluate(X3_test, y_test, batch_size=128,␣
↪verbose=2)
print('Accuracy on test dataset:', test_accuracy)
```

```
4/4 - 0s - loss: 0.0035 - accuracy: 0.9976
Accuracy on test dataset: 0.9976076483726501
```

VGG-16 using only training of the classifier head gives exceptional performance. The validation curves are smooth (unlike the case for the CIFAR-10 style fit), which gives confidence in the result. There is no evidence of overfitting. The accuracy on the test data set is 0.993.

Fine tuning not required since obtained accuracy is sufficient. The next step would be to try fine tuning, i.e., continuing the preceding fit but releasing also the top layer weights in the VGG-16, and using an even smaller learning rate. However, this is seen to be not necessary.

Conclusions

We found that the CIFAR-10 style CNN (with 35,809 trainable parameters, compared to our training set size of 1,949) did not give trustworthy results with either the larger or smaller learning rate as evidenced by the extreme varibility in the validation accuracy and loss curves. However, a transfer learning approach using the VGG-16 CNN model as the base (Total params: 15,108,417; Trainable params: 393,729) gave exceptionally smooth and good results, with an accuracy on the test data set of 0.998. This again affirms the transfer learning method and its facile and broad applicability across diverse fields.