train_cnn

October 26, 2019

1 Training a Texture Classification CNN on MVTec AD

1.1 Imports

```
In [1]: import os
        import numpy as np
        import tensorflow
        import keras

        from keras.preprocessing.image import ImageDataGenerator
        from keras.models import Sequential
        from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, Dense
        from keras import regularizers
        from keras.models import load_model

        # from tensorflow.python.summary import summary_iterator
        from tensorflow.core.util import event_pb2
        from tensorflow.python.lib.io import tf_record

        import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
```

Find the Python distribution used for the Jupyter notebook:

```
In [2]: import sys
        print(sys.executable)
```

```
C:\Users\chanlynd\Anaconda3\python.exe
```

If Keras/TensorFlow are not yet installed in below Python distribution, run `pip install keras`/`pip install tensorflow`.

1.2 Load data

```
In [3]: # Settings
        DATASET_ROOT = 'data/train'
        CLASSES = ['carpet', 'grid', 'leather', 'tile', 'wood']

        # Read in the data
        train_datagen = ImageDataGenerator(rescale=1./255,
                                           shear_range=0.2,
                                           rotation_range=30,
                                           horizontal_flip=True,
                                           vertical_flip=True)
        train_generator = train_datagen.flow_from_directory(DATASET_ROOT,
                                                            class_mode='categorical',
                                                            interpolation='bilinear',
                                                            target_size=(224, 224),
                                                            batch_size=16,
                                                            shuffle=True,
                                                            classes=CLASSES)
        unique, counts = np.unique(train_generator.labels, return_counts=True)
        print(dict(zip(CLASSES, counts)))
```

```
Found 1266 images belonging to 5 classes.
{'carpet': 280, 'grid': 264, 'leather': 245, 'tile': 230, 'wood': 247}
```

1.3 Define model architecture

```
In [4]: # Read in the model
        weight_decay = 5e-4
        model = Sequential()
        model.add(Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu', input_shape
        model.add(Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu', kernel_regu
        model.add(MaxPooling2D(pool_size=(2, 2)))

        model.add(Conv2D(128, kernel_size=(3, 3), padding='same', activation='relu', kernel_reg
        model.add(Conv2D(128, kernel_size=(3, 3), padding='same', activation='relu', kernel_reg
        model.add(MaxPooling2D(pool_size=(2, 2)))

        model.add(Conv2D(256, kernel_size=(3, 3), padding='same', activation='relu', kernel_reg
        model.add(Conv2D(256, kernel_size=(3, 3), padding='same', activation='relu', kernel_reg
        model.add(Conv2D(256, kernel_size=(3, 3), padding='same', activation='relu', kernel_reg

        model.add(GlobalAveragePooling2D())
        model.add(Dense(len(CLASSES), activation='softmax', kernel_initializer=keras.initialize
                        bias_initializer=keras.initializers.Zeros(), kernel_regularizer=regula

        # Compile the model
        model.compile(loss=keras.losses.categorical_crossentropy,
                      optimizer=keras.optimizers.Adadelta(),
                      metrics=['accuracy'])
```

1.4 Train the model

WARNING: this will train the model for 100 epochs, do not run unless necessary

```
In [ ]: # def lr_scheduler(epoch):
        #     # return 1e-3 * (.5 ** (epoch // 5))
        #     return 1e-2 * (.5 ** (epoch // 20))
        # lr_reduce_cb = keras.callbacks.LearningRateScheduler(lr_scheduler)
        # tensorboard_cb = keras.callbacks.TensorBoard(log_dir='log', write_graph=True)
        # model.fit_generator(generator=train_generator,
        #                     steps_per_epoch=train_generator.n // 16,
        #                     epochs=100,
        #                     callbacks=[lr_reduce_cb, tensorboard_cb],
        #                     verbose=2)
```

1.5 Save the model

WARNING: this will overwrite the saved model provided, do not run unless necessary

```
In [ ]: # if not os.path.exists(save_dir):
        #     os.makedirs(save_dir)
        # model.save('save/texturenet.h5')
```

1.6 Load the saved model

```
In [5]: model = load_model('save/texturenet.h5')
```

1.7 Load training progress from Tensorboard

```
In [6]: def my_summary_iterator(path):
            for r in tf_record.tf_record_iterator(path):
                yield event_pb2.Event.FromString(r)

        train_acc = []
        train_loss = []
        train_lr = []
        print('log/' + os.listdir('log')[0])
        for e in my_summary_iterator('log/' + os.listdir('log')[0]):
            for v in e.summary.value:
                if v.tag == 'acc':
                    train_acc.append(v.simple_value)
                elif v.tag == 'loss':
                    train_loss.append(v.simple_value)
                elif v.tag == 'lr':
                    train_lr.append(v.simple_value)
```

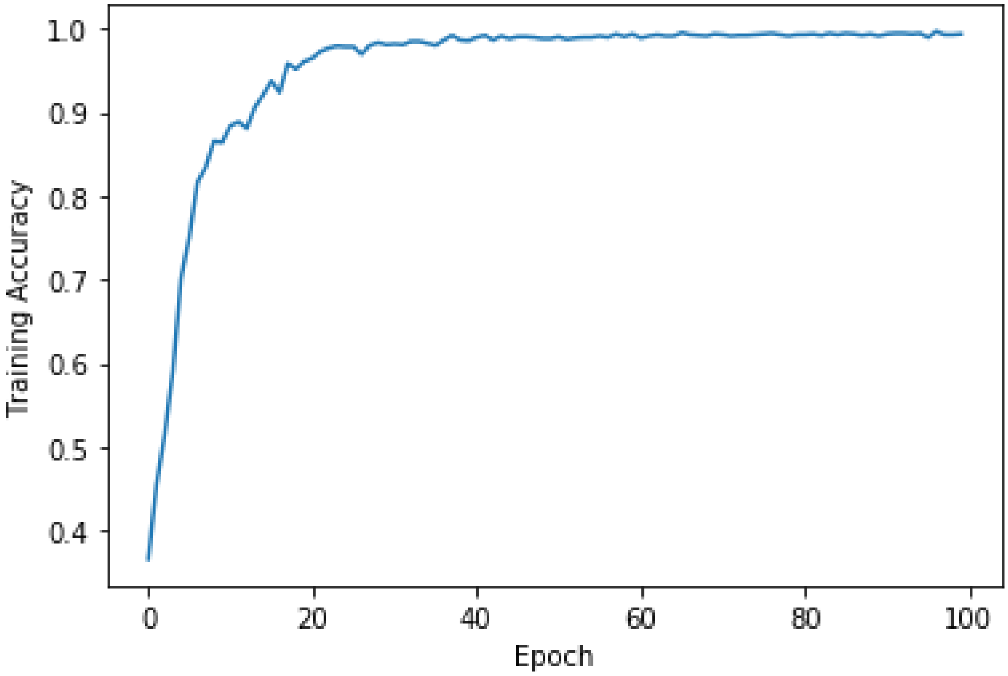
```
log/events.out.tfevents.1571952078.HADES
WARNING:tensorflow:From <ipython-input-6-00d5ea4b2a47>:2: tf_record_iterator (from tensorflow.
Instructions for updating:
```

```
Use eager execution and:
`tf.data.TFRecordDataset(path)`
```

1.8 Plot training accuracy

```
In [7]: plt.plot(np.arange(len(train_acc)), train_acc)
        plt.xlabel('Epoch')
        plt.ylabel('Training Accuracy')
```

```
Out[7]: Text(0, 0.5, 'Training Accuracy')
```

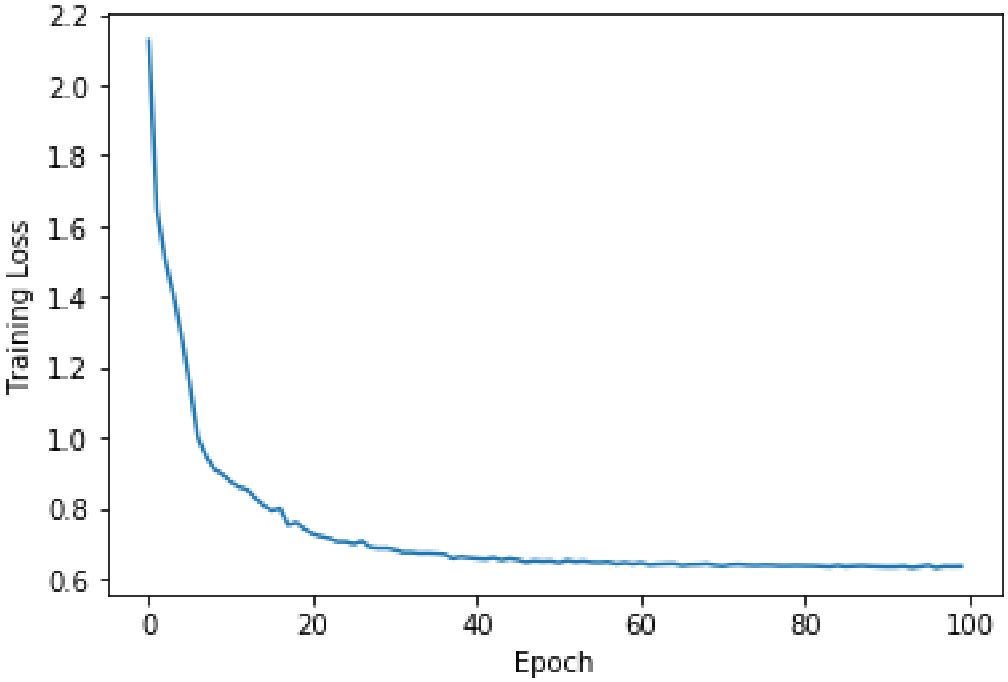


```
In [8]: print('Final training accuracy=%f' % train_acc[-1])
```

```
Final training accuracy=0.993600
```

```
In [9]: plt.plot(np.arange(len(train_loss)), train_loss)
        plt.xlabel('Epoch')
        plt.ylabel('Training Loss')
```

```
Out[9]: Text(0, 0.5, 'Training Loss')
```

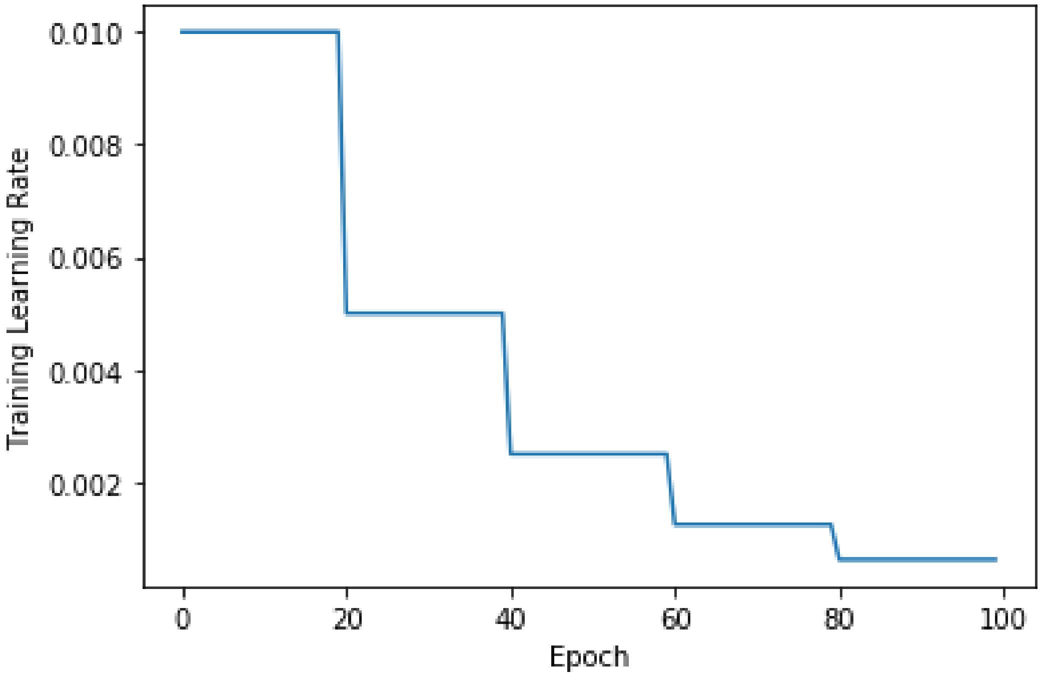


```
In [10]: print('Final training loss=%f' % train_loss[-1])

Final training loss=0.635351
```

```
In [11]: plt.plot(np.arange(len(train_lr)), train_lr)
         plt.xlabel('Epoch')
         plt.ylabel('Training Learning Rate')

Out[11]: Text(0, 0.5, 'Training Learning Rate')
```


0.010
0.008
0.006
0.004
0.002
Training Learning Rate
0
20
40
60
80
100
Epoch

```
In [ ]:
```