Assignment 3

CSCI E-80a Intro to Artificial Intelligence

Student: Steven Devisch
Student ID: 40972382

In this report I will
- add a convolutional (CNN) layer, a max pool layer and a flattening layer to the generic_vns_function developed in assignment 2
- train, test and compare performance with the NN from the previous Dense Layers Assignment for both handwritten digits and speech.
- tinker with batch size, learning rate, the convolutional layer's hyper-parameters (filter size, etc) and epochs to obtain better performance.

I noticed the code provides for early stopping. This maximizes the number of epochs while not continue to running epochs unnecessarily. I left the epochs, therefore at 100.

I also noticed there was a typo in the normalization code. I replaced 'if string is "computer_vision"' with 'if string is "computer_vision"', adding an underscore.

1. Benchmark

I first benchmarked performance of the dense model.

Table 1: Benchmark Validation Accuracy and Validation Loss for vision and speech

	Computer Vision	Speech Recognition
Last epoch	8	22
Val. Accuracy (%)	97.1	69.04
Val. Loss	0.0956	4.0715
Baseline error (%)	2.86	30.96

The model seems to work reasonably well for computer vision, but not for speech recognition.

2. Adding convolutional

I added a convolutional layer.

```
model = models.Sequential(
[
    layers.Input(shape=input_dim),
    layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    # layers.Dropout(0.5),
    layers.Dense(classes, activation="softmax"),
]
)
```

Table 2: Results with convolution

	Computer Vision	Speech Recognition
Last epoch	70	17
Val. Accuracy (%)	98.1	67.16
Val. Loss	0.0574	1.1838
Baseline error (%)	1.86	32.84

After adding convolution, computer vision improves materially, but the speech recognition model is not showing improvement yet. In fact, we see mild deterioration. I will adjust the model formulation and hyper parameters to try to improve the model further

3. Experiment 1: Changing learning rate

The original learning rate was 0.0001. I will now modify the learning rates for all models to see how they behave. In theory, I could just manually change the variable "lr" in my code every time before running the command "python assignment2_answer.py". In order to speed up the process, I modified the code to loop over multiple learning rates at once:

```
learning_rate = [1, 0.1, 0.001, 0.0001, 0.00001]

    # Loop over several learning rates
    for lr in learning_rate:
```

```
        # Generate and train model
        model = generic_vns_function(X_train.shape[1], layers, y_train.shape[1],
                                     layer_units, lr)
        trained_model = train_model(model, epochs, batch_size, X_train, y_train,
                                    X_test, y_test)

        # Save model to h5 file
        trained_model.save('models/model_%s_a2.h5' % dataset)
```

To be able to easily compare my results, I set “verbose” in the training function to 0, in order to avoid printing of progress per epoch, and printed accuracy and loss instead standard error:

```
def train_model(model, epochs, batch_size, x_train, y_train, x_test, y_test):
    """Generic Deep Learning Model training function."""
    cb = [callbacks.EarlyStopping(monitor='val_loss', patience=3)]
    model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=epochs,
              batch_size=batch_size, verbose=0, callbacks=cb)
    scores = model.evaluate(x_test, y_test, verbose=2)

    print("Baseline Error: %.2f%%" % (100-scores[1]*100))
```

I obtain the following results.

		Computer Vision	Speech Recognition
lr = 1	Val. Accuracy (%)	11.35	9.76
	Val. Loss	2.3357	2.3555
lr = 0.1	Val. Accuracy (%)	95.3	24.76
	Val. Loss	0.0790	2.6509
lr = 0.001	Val. Accuracy (%)	**98.38**	**72.06**
	Val. Loss	**0.0535**	**0.8552**
lr = 0.0001	Val. Accuracy (%)	*98.32*	67.91
	Val. Loss	0.0277	0.9643
lr = 0.00001	Val. Accuracy (%)	94.85	65.52
	Val. Loss	0.1209	1.0155

I didn’t include the learning rate = 0.000001 as compute times go up, and as we already saw the model worsening at lr = 0.001.
BEST LEARNING RATE: 0.001

3. Experiment 2: Adding a second convolution Layer

I will now compare one to two convolutional layers

		Computer Vision	Speech Recognition
# Layers = 1	Val. Accuracy (%)	98.38	72.06
	Val. Loss	0.0535	0.8552
# Layers = 2	Val. Accuracy (%)	**98.19**	**83.74**
	Val. Loss	**0.0308**	**0.5460**

Adding a second convolutional layer improved both models.

```
@Secure_Voice_Channel
def generic_vns_function(input_dim, number_dense_layers, classes, lr):
    """Generic Deep Learning Model generator."""
    model = models.Sequential(
    [
        keras.Input(shape=input_dim),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(classes, activation="softmax"),
    ]
    )
    opt = Adam(lr=lr)
    model.compile(loss='categorical_crossentropy', optimizer=opt,
                  metrics=['accuracy'])
    return model
```

Best # Convolutional Layers = 2

5. Experiment 4: Batch size

I first decreased batch size to 100 and observed improved performance. I subsequently ran an additional experiment with batch size 50.

		Computer Vision	Speech Recognition
Batch Size = 50	Val. Accuracy (%)	**99.10**	**87.38**
	Val. Loss	**0.0131**	**0.3026**
Batch Size = 100	Val. Accuracy (%)	99.24	86.01
	Val. Loss	0.0239	0.3408
Batch Size = 200	Val. Accuracy (%)	98.19	83.74
	Val. Loss	0.0308	0.5460

Reducing the batch size primarily benefits the speech recognition model.

BEST batch size = 50

6. Final Model

The code below represents my final optimized model:

```
@Secure_Voice_Channel
def generic_vns_function(input_dim, number_dense_layers, classes, lr):
    """Generic Deep Learning Model generator."""
    model = models.Sequential(
    [
        keras.Input(shape=input_dim),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(classes, activation="softmax"),
    ]
    )
    opt = Adam(lr=lr)
    model.compile(loss='categorical_crossentropy', optimizer=opt,
                  metrics=['accuracy'])
    return model
```

```
def main():

    # Hyperparameters
    layers = 1
    # layer_units = 1000
    epochs = 100
    batch_size = 50
    #lr = 0.0001
    #learning_rate = [1, 0.1, 0.001, 0.0001, 0.00001]
    learning_rate = [0.001]

    # Dataset : "computer_vision" or "speech_recognition"
    dataset = "speech_recognition"
    # Import Datasets

    # printing basic info about the dataset
    # Model / data parameters
    (x_train, y_train), (x_test, y_test) = choose_dataset(dataset)

    print("dataset with load data:", dataset)
    print("x_train shape:", x_train.shape)
    print("y_train shape:", y_train.shape)
    print(x_train.shape[0], "train samples")
    print(x_test.shape[0], "test samples")

    # Generate and train model
    # TODO: Change inputs to generic_vns_function
    # model = generic_vns_function(x_train.shape[1], layers, y_train.shape[1], layer_units, lr)

    nr_of_classes = y_train.shape[1]
    # Loop over several learning rates
    for lr in learning_rate:
        # Generate and train model
        model = generic_vns_function((x_train.shape[1], x_train.shape[2],
                                      x_train.shape[3]), layers, nr_of_classes, lr)
        # def generic_vns_function(input_dim, number_dense_layers, classes, units, lr):
        trained_model = train_model(model, epochs, batch_size, x_train,
                                    y_train, x_test, y_test)

        # Save model to h5 file
        trained_model.save('models/model_%s_a3.h5' % dataset)

    return None
```

I attach a the folder “models” with files for each model with best accuracies:

- For Speech Recognition model_speech_recognition_a3.h5
- For Computer Vision model_computer_vision_a3.h5
-

The table below shows accuracies and losses for all models:

	Computer Vision	Speech Recognition
Epochs	14	22
Orig. Val. Accuracy (%)	97.1	69.04
Orig. Val. Loss	0.0956	4.0715
Improved Val. Accuracy (%)	**99.10**	**87.38**
Improvement Val. Loss	**0.0131**	**0.3026**
Improvement over benchmark acc.	+ 2	+ 18.34
Val accuracy was improved by	69%	59%

Convolutional networks improved model accuracy significantly. On average, validation error (100%- val. accuracy) was reduced by 64%.

7. ANNEX: Command history

In this command history, I show how I run my model. During the assignment, I had to repeat this process many more times, as I manipulated the different hyperparameters.

```
(base) STEVENs-MacBook-Pro:Week03 stevendevisch$ conda activate deep37
(deep37) STEVENs-MacBook-Pro:Week03 stevendevisch$ python assignment3_4.py
/Users/stevendevisch/E80a
dataset with load data: computer_vision
x_train shape: (60000, 28, 28, 1)
y_train shape: (60000, 10)
60000 train samples
10000 test samples
Established Secure Connection.
2020-09-20 12:18:55.026730: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU
 supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
2020-09-20 12:18:55.044412: I tensorflow/compiler/xla/service/service.cc:168] XLA service
 0x7ffb20f3d740 executing computations on platform Host. Devices:
2020-09-20 12:18:55.044434: I tensorflow/compiler/xla/service/service.cc:175]   StreamExe
cutor device (0): Host, Default Version
Ended Secure Connection.
10000/1 - 1s - loss: 2.3317 - accuracy: 0.0982
Baseline Error: 90.18%
Accuracy: 9.82%
Loss: 2.3445
```