FCN-FFA-20E-16L-01

March 24, 2021

1 Are Relations Relevant in CNNs? *A Study Based on a Facial Dataset*

1.1 FCN with Features Further Apart *(20 Epochs - 16 Layers)*

1.1.1 Imports, Seed, GPU integration

```python
[1]: import numpy as np
     import random
     import tensorflow as tf
```

```python
[2]: # Seeds for better reproducibility
     seed = 42
     np.random.seed(seed)
     random.seed(seed)
     tf.random.set_seed(seed)
```

```python
[3]: from tensorflow.keras.layers import  Dropout, Conv2D, GlobalMaxPooling2D,␣
      ↪BatchNormalization, Activation, MaxPool2D
     from tensorflow.keras.optimizers import Adam
     from tensorflow.keras.models import load_model
     from tensorflow.keras.preprocessing.image import ImageDataGenerator
     from tensorflow.python.keras.models import Sequential
     from sklearn.metrics import confusion_matrix
     import itertools
     import matplotlib.pyplot as plt
     import warnings
     warnings.simplefilter(action='ignore', category=FutureWarning)
     %matplotlib inline
```

```python
[4]: physical_devices = tf.config.experimental.list_physical_devices('GPU')
     print("Num GPUs Available: ", len(physical_devices))
     tf.config.experimental.set_memory_growth(physical_devices[0], True)
```

```
Num GPUs Available:  1
```

1.1.2 Data preparation

```python
train_path = '../../picasso_dataset/FFA-data/middle/train'
valid_path = '../../picasso_dataset/FFA-data/middle/valid'
test_path = '../../picasso_dataset/FFA-data/middle/test'
```

```python
train_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.
 ↪vgg16.preprocess_input) \
    .flow_from_directory(directory=train_path, target_size=(224,224),⊔
 ↪classes=['no_face', 'face'], batch_size=20)

valid_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.
 ↪vgg16.preprocess_input) \
    .flow_from_directory(directory=valid_path, target_size=(224,224),⊔
 ↪classes=['no_face', 'face'], batch_size=10)

test_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.
 ↪vgg16.preprocess_input) \
    .flow_from_directory(directory=test_path, target_size=(224,224),⊔
 ↪classes=['no_face', 'face'], batch_size=10, shuffle=False)
```

```
Found 16002 images belonging to 2 classes.
Found 998 images belonging to 2 classes.
Found 3000 images belonging to 2 classes.
```

```python
assert train_batches.n == 16002
assert valid_batches.n == 998
assert test_batches.n == 3000
assert train_batches.num_classes == valid_batches.num_classes == test_batches.
 ↪num_classes == 2
```

1.1.3 Building and training the FCN

```python
dropout_rate=0.2
```

```python
model = Sequential(name = "FCN-FFA")

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding =⊔
 ↪'same', input_shape=(224,224,3), name = "Conv_1"))
model.add(MaxPool2D(pool_size=(2, 2), name = "Max_1"))
model.add(Dropout(rate=dropout_rate, name = "DO_1"))
model.add(BatchNormalization(name = "BN_1"))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding =⊔
 ↪'same', name = "Conv_2"))
model.add(MaxPool2D(pool_size=(2, 2), name = "Max_2"))
```

```python
model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding =
 ↪'same', name = "Conv_3"))
model.add(MaxPool2D(pool_size=(2, 2), name = "Max_3"))
model.add(Dropout(rate=dropout_rate, name = "DO_3"))

model.add(Conv2D(filters=256, kernel_size=(3, 3), activation='relu', padding =
 ↪'same', name = "Conv_4"))
model.add(MaxPool2D(pool_size=(2, 2), name = "Max_4"))

model.add(Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding =
 ↪'same', name = "Conv_5"))
model.add(MaxPool2D(pool_size=(2, 2), name = "Max_5"))

# Fully connected layer
model.add(Conv2D(filters=2, kernel_size=(1,1), name = "Conv_con"))
model.add(GlobalMaxPooling2D(name = "GMax_con"))
model.add(Activation('softmax', name = "Act_con"))

model.summary()
untrained_weights = list(model.get_weights()[0][0][0][0])
```

Model: "FCN-FFA"

Layer (type)	Output Shape	Param #
Conv_1 (Conv2D)	(None, 224, 224, 32)	896
Max_1 (MaxPooling2D)	(None, 112, 112, 32)	0
DO_1 (Dropout)	(None, 112, 112, 32)	0
BN_1 (BatchNormalization)	(None, 112, 112, 32)	128
Conv_2 (Conv2D)	(None, 112, 112, 64)	18496
Max_2 (MaxPooling2D)	(None, 56, 56, 64)	0
Conv_3 (Conv2D)	(None, 56, 56, 128)	73856
Max_3 (MaxPooling2D)	(None, 28, 28, 128)	0
DO_3 (Dropout)	(None, 28, 28, 128)	0
Conv_4 (Conv2D)	(None, 28, 28, 256)	295168
Max_4 (MaxPooling2D)	(None, 14, 14, 256)	0

```
Conv_5 (Conv2D)              (None, 14, 14, 512)      1180160
_____
Max_5 (MaxPooling2D)         (None, 7, 7, 512)        0
_____
Conv_con (Conv2D)            (None, 7, 7, 2)          1026
_____
GMax_con (GlobalMaxPooling2D (None, 2)                0
_____
Act_con (Activation)         (None, 2)                0
=================================================================
Total params: 1,569,730
Trainable params: 1,569,666
Non-trainable params: 64
_____
```

```python
[10]: model.compile(optimizer=Adam(learning_rate=0.0001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'] )
```

```python
[11]: history = model.fit(x=train_batches,
              steps_per_epoch=len(train_batches),
              validation_data=valid_batches,
              validation_steps=len(valid_batches),
              epochs=20,
              verbose=2 )
```

```
Epoch 1/20
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the
batch time (batch time: 0.0148s vs `on_train_batch_end` time: 0.0241s). Check
your callbacks.
801/801 - 43s - loss: 0.2466 - accuracy: 0.8670 - val_loss: 0.0651 -
val_accuracy: 0.9820
Epoch 2/20
801/801 - 43s - loss: 0.0240 - accuracy: 0.9922 - val_loss: 0.0226 -
val_accuracy: 0.9940
Epoch 3/20
801/801 - 43s - loss: 0.0095 - accuracy: 0.9970 - val_loss: 0.0141 -
val_accuracy: 0.9950
Epoch 4/20
801/801 - 43s - loss: 0.0072 - accuracy: 0.9977 - val_loss: 0.0032 -
val_accuracy: 0.9990
Epoch 5/20
801/801 - 43s - loss: 0.0013 - accuracy: 0.9996 - val_loss: 0.0022 -
val_accuracy: 1.0000
Epoch 6/20
801/801 - 43s - loss: 0.0017 - accuracy: 0.9994 - val_loss: 0.0151 -
val_accuracy: 0.9930
Epoch 7/20
```

```
801/801 - 43s - loss: 0.0044 - accuracy: 0.9986 - val_loss: 0.0068 -
val_accuracy: 0.9970
Epoch 8/20
801/801 - 44s - loss: 1.5352e-04 - accuracy: 1.0000 - val_loss: 6.1758e-04 -
val_accuracy: 1.0000
Epoch 9/20
801/801 - 43s - loss: 7.7580e-05 - accuracy: 1.0000 - val_loss: 4.8220e-04 -
val_accuracy: 1.0000
Epoch 10/20
801/801 - 43s - loss: 0.0060 - accuracy: 0.9979 - val_loss: 3.1798e-04 -
val_accuracy: 1.0000
Epoch 11/20
801/801 - 43s - loss: 0.0011 - accuracy: 0.9996 - val_loss: 9.2989e-05 -
val_accuracy: 1.0000
Epoch 12/20
801/801 - 43s - loss: 8.7155e-05 - accuracy: 1.0000 - val_loss: 0.0013 -
val_accuracy: 0.9990
Epoch 13/20
801/801 - 43s - loss: 0.0041 - accuracy: 0.9991 - val_loss: 0.0020 -
val_accuracy: 1.0000
Epoch 14/20
801/801 - 43s - loss: 4.1547e-04 - accuracy: 0.9998 - val_loss: 5.6499e-04 -
val_accuracy: 1.0000
Epoch 15/20
801/801 - 43s - loss: 3.5314e-05 - accuracy: 1.0000 - val_loss: 3.7515e-05 -
val_accuracy: 1.0000
Epoch 16/20
801/801 - 43s - loss: 0.0055 - accuracy: 0.9986 - val_loss: 0.0019 -
val_accuracy: 0.9990
Epoch 17/20
801/801 - 43s - loss: 1.0420e-04 - accuracy: 1.0000 - val_loss: 5.4075e-05 -
val_accuracy: 1.0000
Epoch 18/20
801/801 - 43s - loss: 9.2891e-05 - accuracy: 1.0000 - val_loss: 1.1856e-04 -
val_accuracy: 1.0000
Epoch 19/20
801/801 - 43s - loss: 2.2212e-05 - accuracy: 1.0000 - val_loss: 2.1921e-05 -
val_accuracy: 1.0000
Epoch 20/20
801/801 - 43s - loss: 5.9461e-06 - accuracy: 1.0000 - val_loss: 1.7220e-05 -
val_accuracy: 1.0000
```

1.1.4 Saving the model

```
[12]: filename='models/FCN-FFA-20E-16L-01.h5'
```

```
[13]: model.save(filename)
      saved_weights = list(model.get_weights()[0][0][0][0])
```

1.1.5 Loading the saved model

```
[14]: loaded_model = load_model(filename)
      loaded_weights = list(loaded_model.get_weights()[0][0][0][0])
```

```
[15]: # Assertion that the model was saved and loaded successfully
      assert untrained_weights != saved_weights
      assert saved_weights == loaded_weights
```

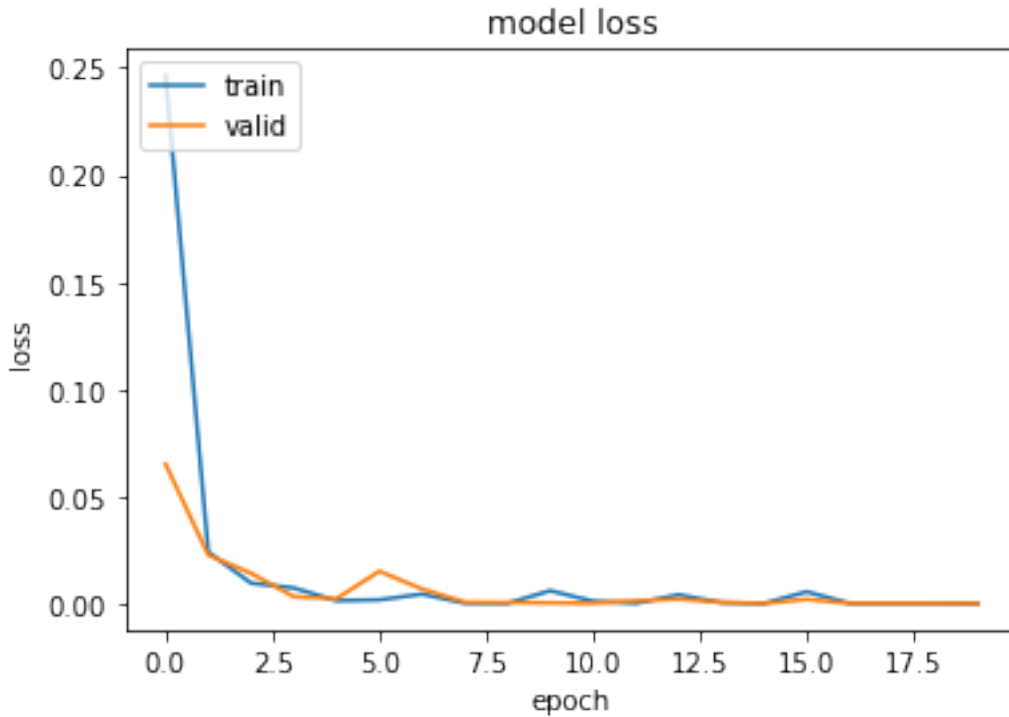### 1.1.6 Accuracy and loss of the trained model

```
[16]: scores = loaded_model.evaluate(test_batches, verbose=2)
      print("Accuracy: %.2f%%" % (scores[1]*100))
      print("Loss: %.2f%%" % (scores[0]*100))
```

```
300/300 - 7s - loss: 2.5875e-05 - accuracy: 1.0000
Accuracy: 100.00%
Loss: 0.00%
```

```
[17]: #Course of accuracy
      plt.plot(history.history['accuracy'])
      plt.plot(history.history['val_accuracy'])
      plt.title('model accuracy')
      plt.ylabel('accuracy')
      plt.xlabel('epoch')
      plt.legend(['train', 'valid'], loc='upper left')
      plt.show()
```

model accuracy
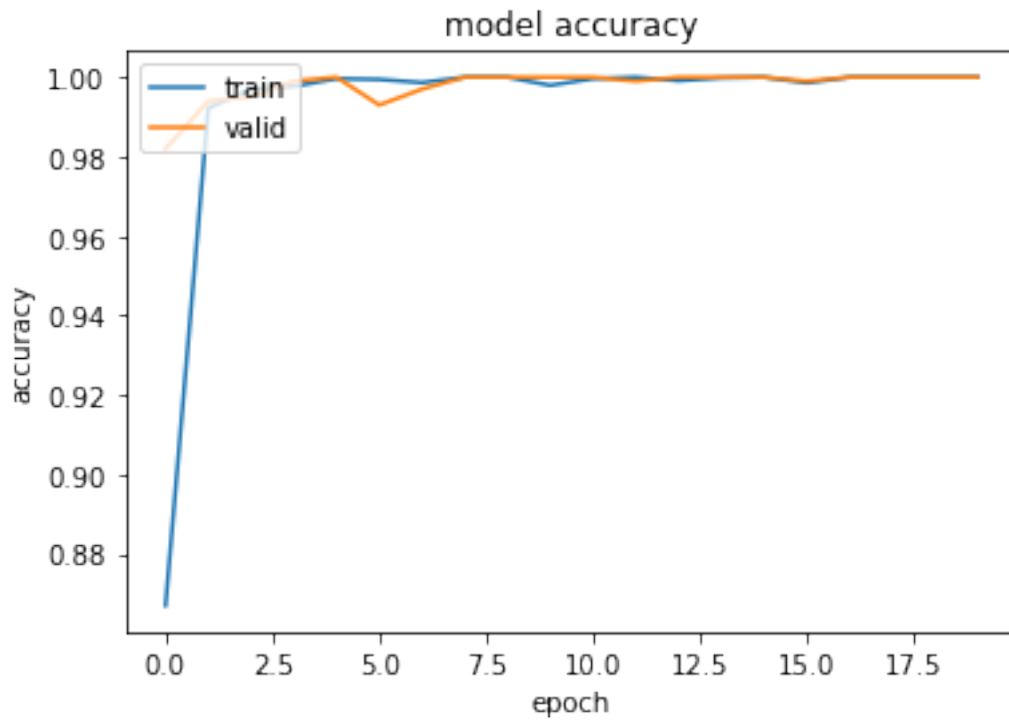
[18]: ```
#Course of loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()
```

### 1.1.7 Testing the CNN

```
[19]: predictions = loaded_model.predict(x=test_batches, steps=len(test_batches),␣
      ↪verbose=0)
```

1.1.8 Index of wrongly predicted pictures

```
[20]: y_true=test_batches.classes
      y_pred=np.argmax(predictions, axis=-1)
      cm = confusion_matrix(y_true = y_true, y_pred = y_pred)
```

```
[21]: face_but_predicted_no_face=[]
      no_face_but_predicted_face=[]

      for i in range(len(predictions)):
              if y_true[i] != y_pred[i]:
                  if y_true[i] == 1:
                      face_but_predicted_no_face.append(i+8001-1500) #Index of file␣
      ↪on disk
                  else:
                      no_face_but_predicted_face.append(i+8001) #Index of file on disk
```

```python
print("Data from class 'face', that was wrongly predicted as 'no-face' [",
 →len(face_but_predicted_no_face), "] :")
print(face_but_predicted_no_face)
print("-----------------------------------------------------------------------------
print("Data from class 'no-face', that was wrongly predicted as 'face' [",
 →len(no_face_but_predicted_face), "] :")
print(no_face_but_predicted_face)
```

```
Data from class 'face', that was wrongly predicted as 'no-face' [ 0 ] :
[]
--------------------------------------------------------------------------------
--------------
Data from class 'no-face', that was wrongly predicted as 'face' [ 0 ] :
[]
```

1.1.9 Confusion matrix

```python
[22]: def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                    horizontalalignment="center",
                    color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
[23]: test_batches.class_indices
```

```
[23]: {'no_face': 0, 'face': 1}
```

```
[24]: cm_plot_labels = ['no_face','face']
      plot_confusion_matrix(cm=cm, classes=cm_plot_labels, title='Confusion Matrix')
```

```
Confusion matrix, without normalization
[[1500    0]
 [   0 1500]]
```

Confusion Matrix

	no_face	face
no_face	1500	0
face	0	1500

True label (vertical axis)
Predicted label (horizontal axis)

```
[ ]:
```