FCN-FFA-30E-14L-02

March 24, 2021

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

1.1 FCN with Features Further Apart *(30 Epochs - 14 Layers)*

1.1.1 Imports, Seed, GPU integration

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

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

[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
```

[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

[5]:
```
train_path = '../../picasso_dataset/FFA-data/middle/train'
valid_path = '../../picasso_dataset/FFA-data/middle/valid'
test_path = '../../picasso_dataset/FFA-data/middle/test'
```

[6]:
```
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.
```

[7]:
```
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

[8]:
```
dropout_rate=0.2
```

[9]:
```
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"))
```

```
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"))

# 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_con (Conv2D)            (None, 14, 14, 2)         514
_________________________________________________________________
GMax_con (GlobalMaxPooling2D (None, 2)                 0
_________________________________________________________________
```

```
Act_con (Activation)         (None, 2)                 0
=================================================================
Total params: 389,058
Trainable params: 388,994
Non-trainable params: 64
_________________________________________________________________
```

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

[11]:
```
history = model.fit(x=train_batches,
          steps_per_epoch=len(train_batches),
          validation_data=valid_batches,
          validation_steps=len(valid_batches),
          epochs=30,
          verbose=2 )
```

```
Epoch 1/30
801/801 - 42s - loss: 0.6413 - accuracy: 0.6090 - val_loss: 0.5274 -
val_accuracy: 0.7375
Epoch 2/30
801/801 - 42s - loss: 0.4342 - accuracy: 0.8047 - val_loss: 0.3782 -
val_accuracy: 0.8467
Epoch 3/30
801/801 - 42s - loss: 0.3247 - accuracy: 0.8623 - val_loss: 0.2995 -
val_accuracy: 0.8838
Epoch 4/30
801/801 - 42s - loss: 0.2509 - accuracy: 0.8984 - val_loss: 0.2543 -
val_accuracy: 0.8928
Epoch 5/30
801/801 - 42s - loss: 0.1827 - accuracy: 0.9288 - val_loss: 0.1813 -
val_accuracy: 0.9188
Epoch 6/30
801/801 - 42s - loss: 0.1418 - accuracy: 0.9454 - val_loss: 0.1369 -
val_accuracy: 0.9519
Epoch 7/30
801/801 - 42s - loss: 0.1132 - accuracy: 0.9578 - val_loss: 0.1073 -
val_accuracy: 0.9649
Epoch 8/30
801/801 - 42s - loss: 0.0855 - accuracy: 0.9694 - val_loss: 0.0669 -
val_accuracy: 0.9810
Epoch 9/30
801/801 - 42s - loss: 0.0660 - accuracy: 0.9779 - val_loss: 0.0627 -
val_accuracy: 0.9790
Epoch 10/30
801/801 - 42s - loss: 0.0513 - accuracy: 0.9826 - val_loss: 0.0423 -
val_accuracy: 0.9870
```

```
Epoch 11/30
801/801 - 42s - loss: 0.0426 - accuracy: 0.9853 - val_loss: 0.0552 -
val_accuracy: 0.9840
Epoch 12/30
801/801 - 42s - loss: 0.0370 - accuracy: 0.9870 - val_loss: 0.0397 -
val_accuracy: 0.9860
Epoch 13/30
801/801 - 42s - loss: 0.0340 - accuracy: 0.9887 - val_loss: 0.0241 -
val_accuracy: 0.9960
Epoch 14/30
801/801 - 42s - loss: 0.0319 - accuracy: 0.9884 - val_loss: 0.0268 -
val_accuracy: 0.9930
Epoch 15/30
801/801 - 42s - loss: 0.0275 - accuracy: 0.9901 - val_loss: 0.0371 -
val_accuracy: 0.9900
Epoch 16/30
801/801 - 42s - loss: 0.0223 - accuracy: 0.9928 - val_loss: 0.0215 -
val_accuracy: 0.9950
Epoch 17/30
801/801 - 42s - loss: 0.0231 - accuracy: 0.9920 - val_loss: 0.0193 -
val_accuracy: 0.9950
Epoch 18/30
801/801 - 42s - loss: 0.0210 - accuracy: 0.9928 - val_loss: 0.0250 -
val_accuracy: 0.9910
Epoch 19/30
801/801 - 42s - loss: 0.0217 - accuracy: 0.9921 - val_loss: 0.0250 -
val_accuracy: 0.9940
Epoch 20/30
801/801 - 42s - loss: 0.0168 - accuracy: 0.9947 - val_loss: 0.0262 -
val_accuracy: 0.9890
Epoch 21/30
801/801 - 42s - loss: 0.0198 - accuracy: 0.9930 - val_loss: 0.0130 -
val_accuracy: 0.9940
Epoch 22/30
801/801 - 42s - loss: 0.0184 - accuracy: 0.9942 - val_loss: 0.0211 -
val_accuracy: 0.9960
Epoch 23/30
801/801 - 42s - loss: 0.0142 - accuracy: 0.9955 - val_loss: 0.0165 -
val_accuracy: 0.9970
Epoch 24/30
801/801 - 42s - loss: 0.0148 - accuracy: 0.9949 - val_loss: 0.0215 -
val_accuracy: 0.9940
Epoch 25/30
801/801 - 42s - loss: 0.0174 - accuracy: 0.9944 - val_loss: 0.0276 -
val_accuracy: 0.9920
Epoch 26/30
801/801 - 42s - loss: 0.0151 - accuracy: 0.9946 - val_loss: 0.0174 -
val_accuracy: 0.9960
```

```
Epoch 27/30
801/801 - 42s - loss: 0.0126 - accuracy: 0.9958 - val_loss: 0.0256 -
val_accuracy: 0.9870
Epoch 28/30
801/801 - 42s - loss: 0.0150 - accuracy: 0.9944 - val_loss: 0.0132 -
val_accuracy: 0.9950
Epoch 29/30
801/801 - 42s - loss: 0.0162 - accuracy: 0.9947 - val_loss: 0.0172 -
val_accuracy: 0.9930
Epoch 30/30
801/801 - 42s - loss: 0.0124 - accuracy: 0.9960 - val_loss: 0.0203 -
val_accuracy: 0.9950
```

1.1.4 Saving the model

[12]:
```
filename='models/FCN-FFA-30E-14L-02.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])
```

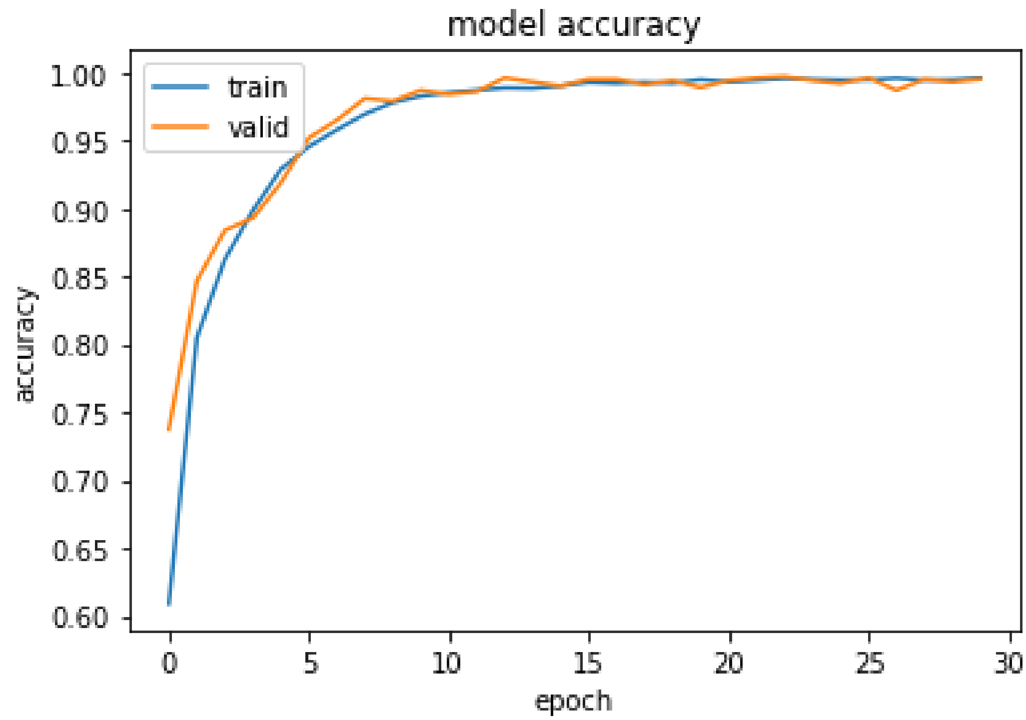
[15]:
```
# Assertion that the model was saved and loaded successfully
assert untrained_weights != saved_weights
assert saved_weights == loaded_weights
```

1.1.6 Accuracy and loss of the trained model

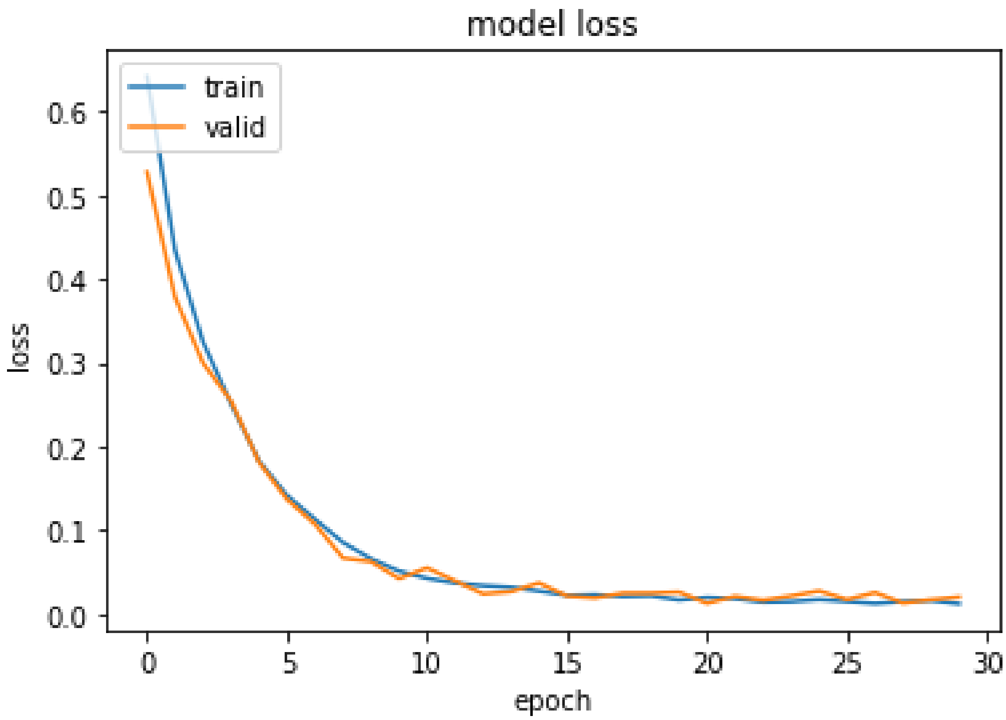
[16]:
```
scores = loaded_model.evaluate(test_batches, verbose=2)
print("Accuracy: %.2f%%" % (scores[1]*100))
print("Loss: %.2f%%" % (scores[0]*100))
```

```
300/300 - 7s - loss: 0.0186 - accuracy: 0.9930
Accuracy: 99.30%
Loss: 1.86%
```

[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()
```



[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
```

```
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' [ 5 ] :
[8400, 8456, 8608, 8773, 9059]
--------------------------------------------------------------------------------
--------------
Data from class 'no-face', that was wrongly predicted as 'face' [ 16 ] :
[8013, 8054, 8193, 8244, 8281, 8365, 8432, 8443, 8451, 8527, 8602, 8850, 8894,
9020, 9090, 9274]
```

1.1.9 Confusion matrix

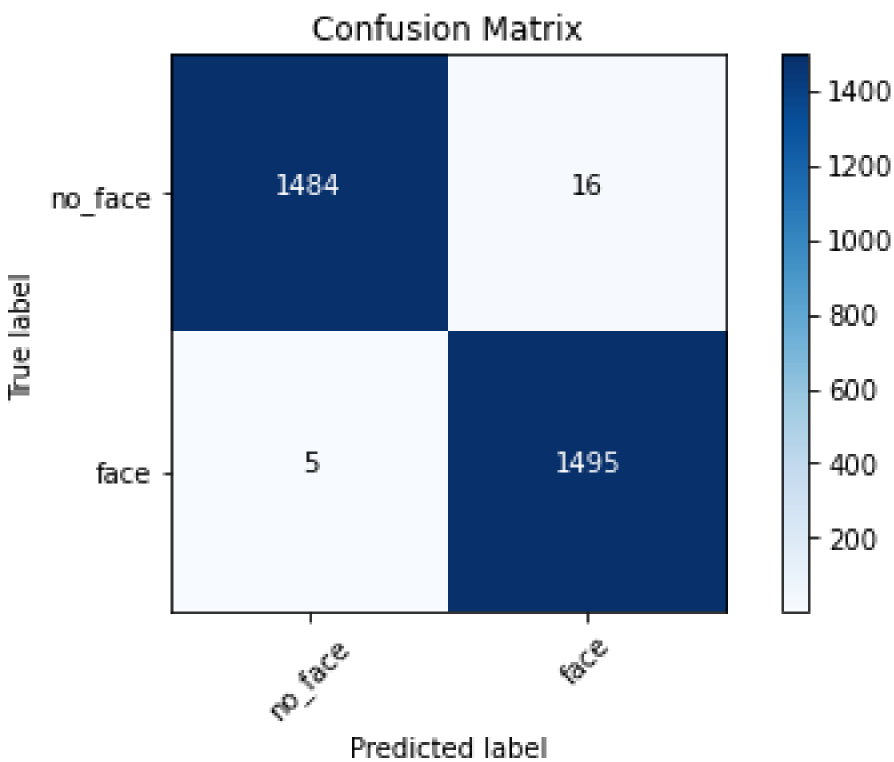
[22]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    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
[[1484   16]
 [   5 1495]]
```



[ ]: