CNN-FFA-30E-13L-02

March 24, 2021

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

1.1 CNN with Features Further Apart *(30 Epochs - 13 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, BatchNormalization,␣
      ↪MaxPool2D, Dense, Flatten
     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 CNN

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

```
[9]: model = Sequential(name = "CNN-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"))

# Fully connected layer
model.add(Flatten(name = "Flat_con"))
model.add(Dense(units=2, activation='softmax', name = "D_con"))

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

```
Model: "CNN-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
_____
Flat_con (Flatten)           (None, 50176)             0
_____
D_con (Dense)                (None, 2)                 100354
=================================================================
Total params: 488,898
```

```
Trainable params: 488,834
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
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the
batch time (batch time: 0.0140s vs `on_train_batch_end` time: 0.0258s). Check
your callbacks.
801/801 - 43s - loss: 0.2164 - accuracy: 0.8957 - val_loss: 0.0751 -
val_accuracy: 0.9699
Epoch 2/30
801/801 - 42s - loss: 0.0282 - accuracy: 0.9899 - val_loss: 0.0386 -
val_accuracy: 0.9850
Epoch 3/30
801/801 - 42s - loss: 0.0117 - accuracy: 0.9959 - val_loss: 0.0361 -
val_accuracy: 0.9880
Epoch 4/30
801/801 - 42s - loss: 0.0057 - accuracy: 0.9983 - val_loss: 0.0043 -
val_accuracy: 0.9980
Epoch 5/30
801/801 - 42s - loss: 0.0032 - accuracy: 0.9991 - val_loss: 0.0057 -
val_accuracy: 0.9980
Epoch 6/30
801/801 - 42s - loss: 0.0021 - accuracy: 0.9992 - val_loss: 0.0041 -
val_accuracy: 0.9990
Epoch 7/30
801/801 - 42s - loss: 0.0010 - accuracy: 0.9997 - val_loss: 0.0031 -
val_accuracy: 0.9980
Epoch 8/30
801/801 - 42s - loss: 0.0033 - accuracy: 0.9989 - val_loss: 0.0025 -
val_accuracy: 0.9980
Epoch 9/30
801/801 - 41s - loss: 5.8469e-04 - accuracy: 0.9999 - val_loss: 5.1549e-04 -
val_accuracy: 1.0000
Epoch 10/30
801/801 - 42s - loss: 0.0017 - accuracy: 0.9992 - val_loss: 8.1835e-04 -
val_accuracy: 1.0000
```

```
Epoch 11/30
801/801 - 42s - loss: 1.3884e-04 - accuracy: 0.9999 - val_loss: 4.5337e-05 -
val_accuracy: 1.0000
Epoch 12/30
801/801 - 42s - loss: 8.5655e-05 - accuracy: 1.0000 - val_loss: 3.5135e-04 -
val_accuracy: 1.0000
Epoch 13/30
801/801 - 42s - loss: 0.0021 - accuracy: 0.9994 - val_loss: 0.0010 -
val_accuracy: 0.9990
Epoch 14/30
801/801 - 42s - loss: 0.0015 - accuracy: 0.9998 - val_loss: 1.4193e-04 -
val_accuracy: 1.0000
Epoch 15/30
801/801 - 42s - loss: 2.0310e-05 - accuracy: 1.0000 - val_loss: 2.2149e-04 -
val_accuracy: 1.0000
Epoch 16/30
801/801 - 41s - loss: 1.4325e-05 - accuracy: 1.0000 - val_loss: 2.3643e-04 -
val_accuracy: 1.0000
Epoch 17/30
801/801 - 42s - loss: 1.8561e-05 - accuracy: 1.0000 - val_loss: 4.2945e-04 -
val_accuracy: 1.0000
Epoch 18/30
801/801 - 42s - loss: 5.8276e-06 - accuracy: 1.0000 - val_loss: 6.6656e-05 -
val_accuracy: 1.0000
Epoch 19/30
801/801 - 42s - loss: 3.1661e-06 - accuracy: 1.0000 - val_loss: 9.9982e-05 -
val_accuracy: 1.0000
Epoch 20/30
801/801 - 42s - loss: 1.3996e-06 - accuracy: 1.0000 - val_loss: 7.8417e-05 -
val_accuracy: 1.0000
Epoch 21/30
801/801 - 42s - loss: 7.3740e-07 - accuracy: 1.0000 - val_loss: 1.1316e-04 -
val_accuracy: 1.0000
Epoch 22/30
801/801 - 42s - loss: 8.3665e-07 - accuracy: 1.0000 - val_loss: 1.2731e-04 -
val_accuracy: 1.0000
Epoch 23/30
801/801 - 42s - loss: 2.2601e-06 - accuracy: 1.0000 - val_loss: 1.8183e-04 -
val_accuracy: 1.0000
Epoch 24/30
801/801 - 42s - loss: 8.8551e-06 - accuracy: 1.0000 - val_loss: 2.4118e-05 -
val_accuracy: 1.0000
Epoch 25/30
801/801 - 42s - loss: 3.1174e-07 - accuracy: 1.0000 - val_loss: 2.9093e-05 -
val_accuracy: 1.0000
Epoch 26/30
801/801 - 42s - loss: 1.9843e-06 - accuracy: 1.0000 - val_loss: 2.3636e-05 -
val_accuracy: 1.0000
```

```
Epoch 27/30
801/801 - 42s - loss: 0.0025 - accuracy: 0.9992 - val_loss: 3.3003e-04 -
val_accuracy: 1.0000
Epoch 28/30
801/801 - 42s - loss: 3.8880e-04 - accuracy: 0.9999 - val_loss: 5.7393e-06 -
val_accuracy: 1.0000
Epoch 29/30
801/801 - 42s - loss: 5.1166e-06 - accuracy: 1.0000 - val_loss: 8.6233e-06 -
val_accuracy: 1.0000
Epoch 30/30
801/801 - 42s - loss: 2.7989e-06 - accuracy: 1.0000 - val_loss: 1.2211e-05 -
val_accuracy: 1.0000
```

1.1.4 Saving the model

```
[12]: filename='models/CNN-FFA-30E-13L-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: 3.7925e-07 - 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
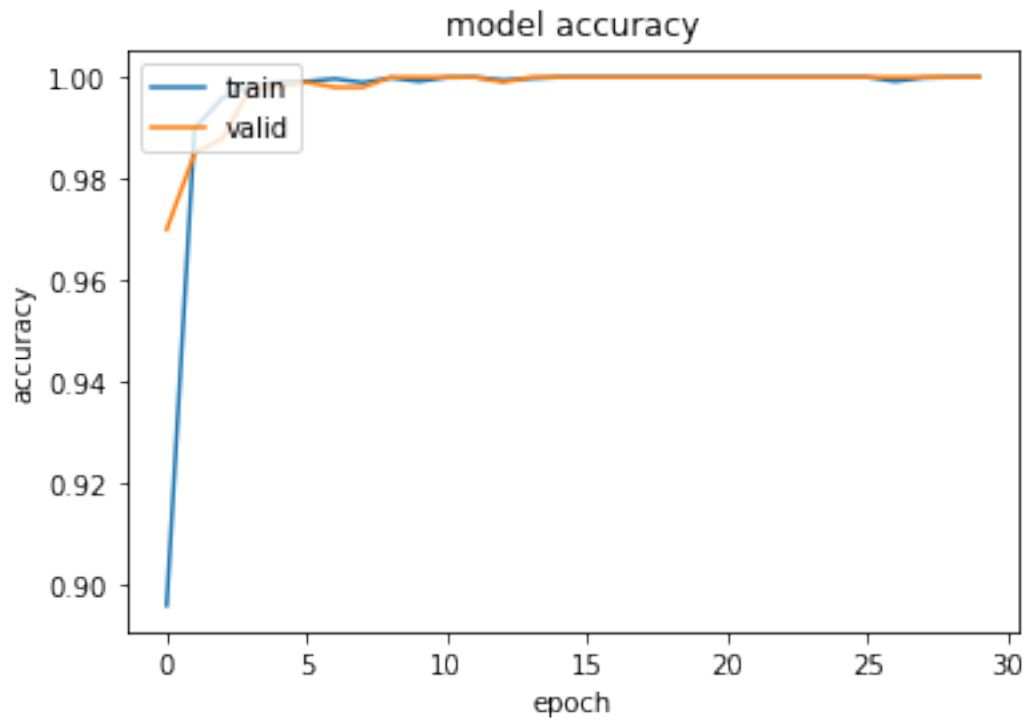
[18]:
```
#Course of loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()
```
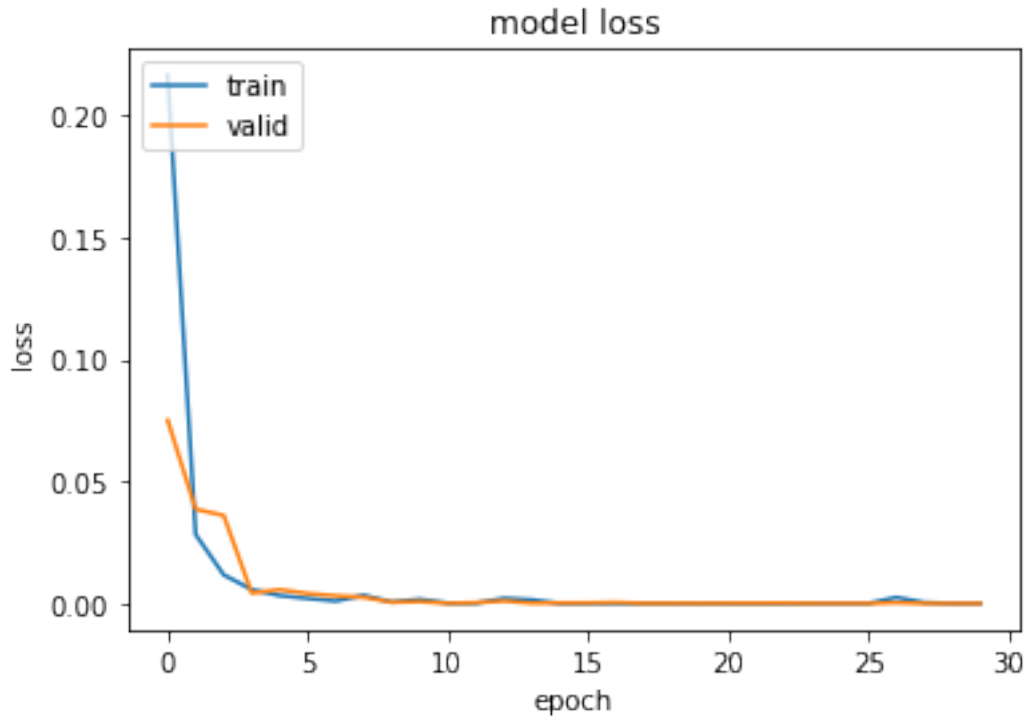
1.1.7 Testing the CNN

```
[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
```

```
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]]
```