Codes and Mathematical Expressions

Codes:

```
DATA_DIR = "../input/data-zel-d2-17-3/data_özel_D2_17_3"
TRAINING_DIR = DATA_DIR + "/train"
VALIDATION_DIR = DATA_DIR + "/val"
TEST_DIR = DATA_DIR + "/test"

BATCH_SIZE=100
EPOCS=200
PATIENCE=100
SHOW_MISTAKEN_PREDICTIONS=True


IMG_WIDTH=150
IMG_HEIGHT=150
```

```
training_datagen =  ImageDataGenerator(rescale = 1./255)
validation_datagen =  ImageDataGenerator(rescale = 1./255)


train_generator = training_datagen.flow_from_directory(
        TRAINING_DIR,
        target_size=(IMG_WIDTH,IMG_HEIGHT),
    batch_size=BATCH_SIZE,
        class_mode='categorical')


validation_generator = validation_datagen.flow_from_directory(
        VALIDATION_DIR,
        target_size=(IMG_WIDTH,IMG_HEIGHT),
    batch_size=BATCH_SIZE,
        class_mode='categorical',
    shuffle=False)

number_of_classes=len(list(train_generator.class_indices.keys()))
model = Sequential([
    # Note the input shape is the desired size of the image 150x150 with 3 byt
es color
    # This is the first convolution
    Conv2D(32, (3,3), activation='relu', input_shape=(IMG_WIDTH,IMG_HEIGHT, 3)
),
    MaxPooling2D(2, 2),
    # The second convolution
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    # The third convolution
    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    # The fourth convolution
    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D(2,2),
```

```python
    # Flatten the results to feed into a DNN
    Flatten(),
    Dropout(0.5),
    # 512 neuron hidden layer
    Dense(512, activation='relu'),
    Dense(number_of_classes, activation='softmax')
])
```

```python
model.compile(loss = 'categorical_crossentropy', optimizer='adam', metrics=['a
ccuracy'])

callbacks = [EarlyStopping(monitor='val_loss', mode='min',patience=PATIENCE),
             ModelCheckpoint(filepath='best_model.h5', monitor='val_loss',mode
='min', save_best_only=True,verbose=1)]


history = model.fit_generator(train_generator,
                              epochs=EPOCS,
                              callbacks=callbacks,
                              validation_data = validation_generator,
                              steps_per_epoch=train_generator.samples // BATCH
_SIZE,
                              validation_steps=validation_generator.samples //
BATCH_SIZE,
                              verbose = 1)
```

```python
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc=0)
plt.figure()


plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

```python
def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):
accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy
```

```
    if cmap is None:
        # Greys, Purples, Blues, Greens, Oranges, Reds, YlOrBr, YlOrRd, OrRd,
PuRd, RdPu, BuPu, GnBu, PuBu, YlGnBu, PuBuGn, BuGn, YlGn
        cmap = plt.get_cmap('YlGn')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=90)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(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\naccuracy={:0.4f}; misclass={:0.4f}'.format(ac
curacy, misclass))
    plt.show()
```

```
def results_for(model_file,
                data_generator,
                caption ):
    model = tf.keras.models.load_model(model_file)
    test_loss, test_acc = model.evaluate_generator(data_generator)
    print(caption,' acc:', test_acc)
    print(caption,' loss:', test_loss)
    # Confusion matrix
    num_of_samples=data_generator.samples
    batch_size=20
    Y_pred = model.predict_generator(data_generator, num_of_samples // batch_s
ize)
    y_pred = np.argmax(Y_pred, axis=1)
    #print(y_pred)
    #print('Confusion Matrix')
    cm=confusion_matrix(data_generator.classes, y_pred)
```

```
    #print(cm)

    target_names=data_generator.class_indices.keys()
    plot_confusion_matrix(cm,normalize = False,target_names=target_names,title
= "Confusion Matrix")
    print('Classification Report')
    print('----------------------')
    print(classification_report(data_generator.classes, y_pred, target_names=t
arget_names))

    # list the mistaken predictions
    if SHOW_MISTAKEN_PREDICTIONS:
        print('\nMistaken predictions')
        print('---------------------')
        filenames=data_generator.filenames
        errors=np.where(y_pred!=data_generator.classes)[0]
        error_count=len(errors)
        for error in errors:
            predicted=list(data_generator.class_indices.keys())[list(data_gene
rator.class_indices.values()).index(y_pred[error])]
            print("Picture:",filenames[error]," Misprediction ==>", predicted)

test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(TEST_DIR,target_size=(150, 1
50),shuffle=False,batch_size=500,class_mode='categorical')

results_for('best_model.h5',test_generator,"test")


results_for('best_model.h5',validation_generator,"validation")
```

Some Mathematical Expressions

The i^{th} feature map is computed as follows:

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{ij}^{(l)} * B_j^{(l-1)}$$

There is a bank of m_1 filters and the output $Y_i^{(l)}$ of the l^{th} layer consists of $m_i^{(l)}$ feature maps of size $m_2^{(l)} x m_3^{(l)}$.

The convolution operation is widely used in digital image processing where the 2D matrix representing the image (I) is convolved with the smaller 2D kernel matrix (K), then the mathematical formulation with zero padding is given:

$$S_{i,j} = (l * K)_{i,j} = \sum_m \sum_n l_{i,j} . K_{i-m,j-n}$$

The operation of the activation function f () is as follows:

$$\emptyset \left(\boldsymbol{Y_i^{(l)}}\right) = \mathrm{f}\left(B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{ij}^{(l)} * Y_j^{(l-1)} \right)$$

ReLU is the most used activation function for convolution layers. It is mathematically defined as:

$$f(x) = \begin{cases} 0 \ if \ x < 0 \\ x \ if \ x \geq 0 \end{cases}$$

Max pooling: Mathematically it has the form:

$$f_{max}(A) = max_{nxm}(A_{nxm})$$