Binary_2_Conv

January 15, 2024

[1]:
```python
import pickle

import numpy as np
from matplotlib import pyplot as plt
from pandas import read_csv
```

[41]:
```python
print("Loading data...")
training_file = './Data/train.p'

sign_names = read_csv("./Data/signname.csv").values[:, 1]

with open(training_file, mode='rb') as f:
    train = pickle.load(f)
images_train, labels_train = train['features'], train['labels']

# Filter only labels 0-8
mask_0_to_8 = labels_train <= 8
images_train_filtered = images_train[mask_0_to_8]
labels_train_filtered = labels_train[mask_0_to_8]
```

Loading data…

[43]:
```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Conv2D, MaxPooling2D,␣
↪Dropout
from sklearn.metrics import f1_score

# Assuming your image dimensions and channels
height = 32  # example height
width = 32   # example width
channels = 3  # RGB channels

# Define a function to calculate F1 score

#def f1_metric(y_true, y_pred):
#    return tf.py_function(f1_score, (y_true, y_pred > 0.5), tf.float64)
```

```
# Define the new model
model = Sequential([
    # First Convolutional Layer with 32 filters, a 3x3 kernel size, 'same'␣
 ↪padding, and ReLU activation
    Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(height,␣
 ↪width, channels)),

    # MaxPooling to downsample the output of the first Convolutional Layer
    MaxPooling2D((2, 2)),

    # Second Convolutional Layer with 64 filters, a 3x3 kernel size, 'same'␣
 ↪padding, and ReLU activation
    Conv2D(64, (3, 3), padding='same', activation='relu'),

    # MaxPooling to downsample the output of the second Convolutional Layer
    MaxPooling2D((2, 2)),

    # Third Convolutional Layer with 128 filters, a 3x3 kernel size, 'same'␣
 ↪padding, and ReLU activation
    Conv2D(128, (3, 3), padding='same', activation='relu'),

    # MaxPooling to downsample the output of the third Convolutional Layer
    MaxPooling2D((2, 2)),

    # Additional Dropout layer after the third Convolutional Layer
    Dropout(0.3),

    # Flatten layer to convert the 2D output of the convolutional layers into a␣
 ↪1D array
    Flatten(),

    # First Dense (fully connected) layer with 128 units and ReLU activation
    Dense(128, activation='relu'),

    # Dropout layer with 50% dropout rate for regularization
    Dropout(0.5),

    # Second Dense layer with 64 units and ReLU activation
    Dense(64, activation='relu'),

    # Output multiclass
    Dense(9, activation='softmax')
])

# Compile the model
```

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy']) # , f1_metric

print("Model compiled")
```

```
Model compiled
```

[45]:
```
validation_file = './Data/valid.p'
print("Load Validation Data")
with open(validation_file, mode='rb') as f:
    valid = pickle.load(f)
images_valid, labels_valid = valid['features'], valid['labels']

mask_0_to_8_valid = labels_valid <= 8
images_valid_filtered = images_valid[mask_0_to_8_valid]
labels_valid_filtered = labels_valid[mask_0_to_8_valid]
```

```
Load Validation Data
```

[45]:
```
Ellipsis
```

[47]:
```
history = model.fit(images_train_filtered, labels_train_filtered, epochs=10,␣
↪validation_split=0.2)
```

```
Epoch 1/10
```

```
2023-12-19 11:32:30.555281: W
external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
28901376 exceeds 10% of free system memory.
```

```
294/294 [==============================] - 97s 315ms/step - loss: 2.3254 -
accuracy: 0.2485 - val_loss: 6.7475 - val_accuracy: 0.0000e+00
Epoch 2/10
294/294 [==============================] - 101s 343ms/step - loss: 1.5829 -
accuracy: 0.3938 - val_loss: 10.0864 - val_accuracy: 0.0162
Epoch 3/10
294/294 [==============================] - 93s 318ms/step - loss: 0.9103 -
accuracy: 0.6654 - val_loss: 11.8935 - val_accuracy: 0.0706
Epoch 4/10
294/294 [==============================] - 91s 311ms/step - loss: 0.5018 -
accuracy: 0.8220 - val_loss: 16.5761 - val_accuracy: 0.1161
Epoch 5/10
294/294 [==============================] - 101s 344ms/step - loss: 0.3213 -
accuracy: 0.8926 - val_loss: 20.4122 - val_accuracy: 0.1327
Epoch 6/10
294/294 [==============================] - 101s 344ms/step - loss: 0.2388 -
accuracy: 0.9253 - val_loss: 24.7720 - val_accuracy: 0.1276
Epoch 7/10
294/294 [==============================] - 89s 302ms/step - loss: 0.1915 -
```

```
accuracy: 0.9380 - val_loss: 23.9175 - val_accuracy: 0.1310
Epoch 8/10
294/294 [==============================] - 89s 304ms/step - loss: 0.1617 -
accuracy: 0.9505 - val_loss: 22.6710 - val_accuracy: 0.1250
Epoch 9/10
294/294 [==============================] - 103s 350ms/step - loss: 0.1585 -
accuracy: 0.9503 - val_loss: 21.7813 - val_accuracy: 0.1348
Epoch 10/10
294/294 [==============================] - 106s 361ms/step - loss: 0.1221 -
accuracy: 0.9614 - val_loss: 26.8246 - val_accuracy: 0.1339
```

[49]:

```python
# Define the file name for saving the model
model_filename = 'Convolution_Model_F1__Ex_2'

# Save the model to a file
model.save(model_filename)
```

```
INFO:tensorflow:Assets written to: Convolution_Model_F1__Ex_2/assets
```

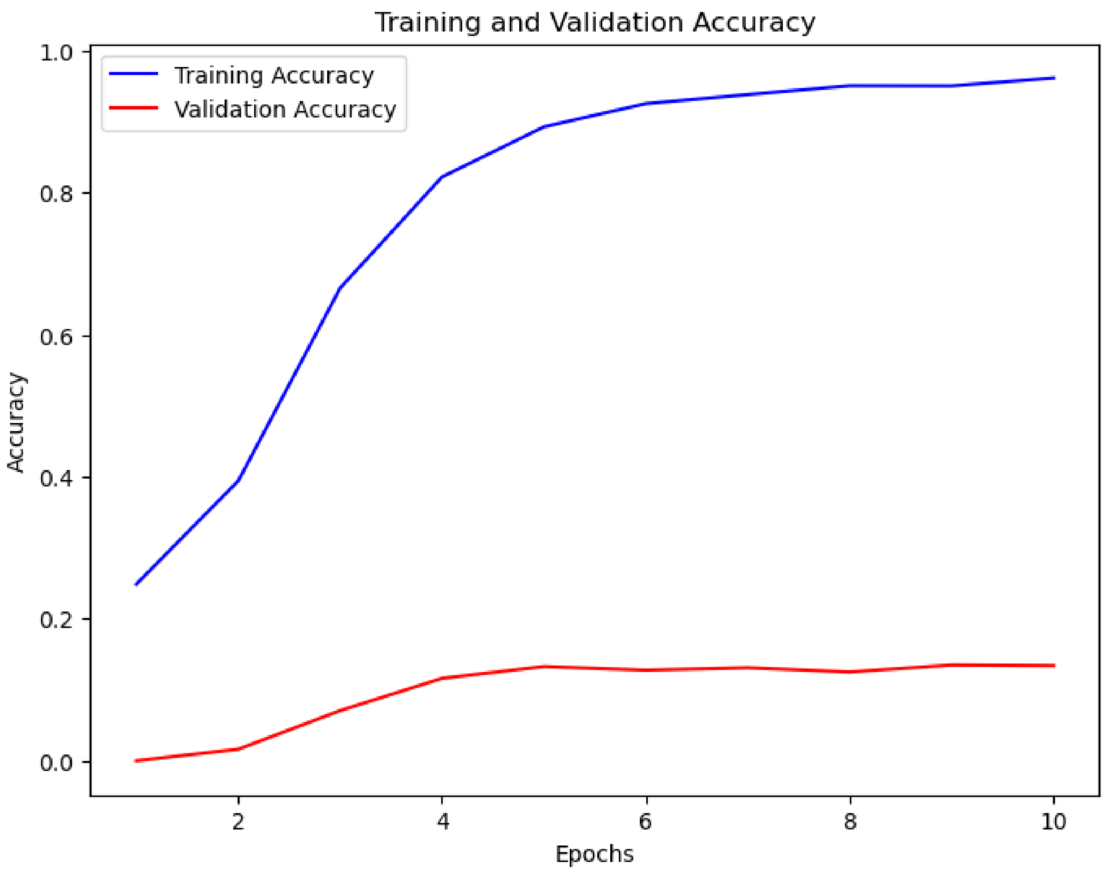
```
INFO:tensorflow:Assets written to: Convolution_Model_F1__Ex_2/assets
```

[51]:

```python
test_file = './Data/test.p'

with open(test_file, mode='rb') as f:
    test = pickle.load(f)
images_test, labels_test = test['features'], test['labels']

mask_test_0_to_8_test = labels_test <= 8
images_test = images_test[mask_test_0_to_8_test]
labels_test = labels_test[mask_test_0_to_8_test]
```

[53]:

```python
# Train the model and store the training history in a variable named "history"

# Extract accuracy values
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

# Plot accuracy
plt.figure(figsize=(8, 6))
epochs = range(1, len(train_accuracy) + 1)
plt.plot(epochs, train_accuracy, 'b', label='Training Accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



[57]:
```
test_loss, test_accuracy  = model.evaluate(images_test, labels_test) # ,f1_score
```

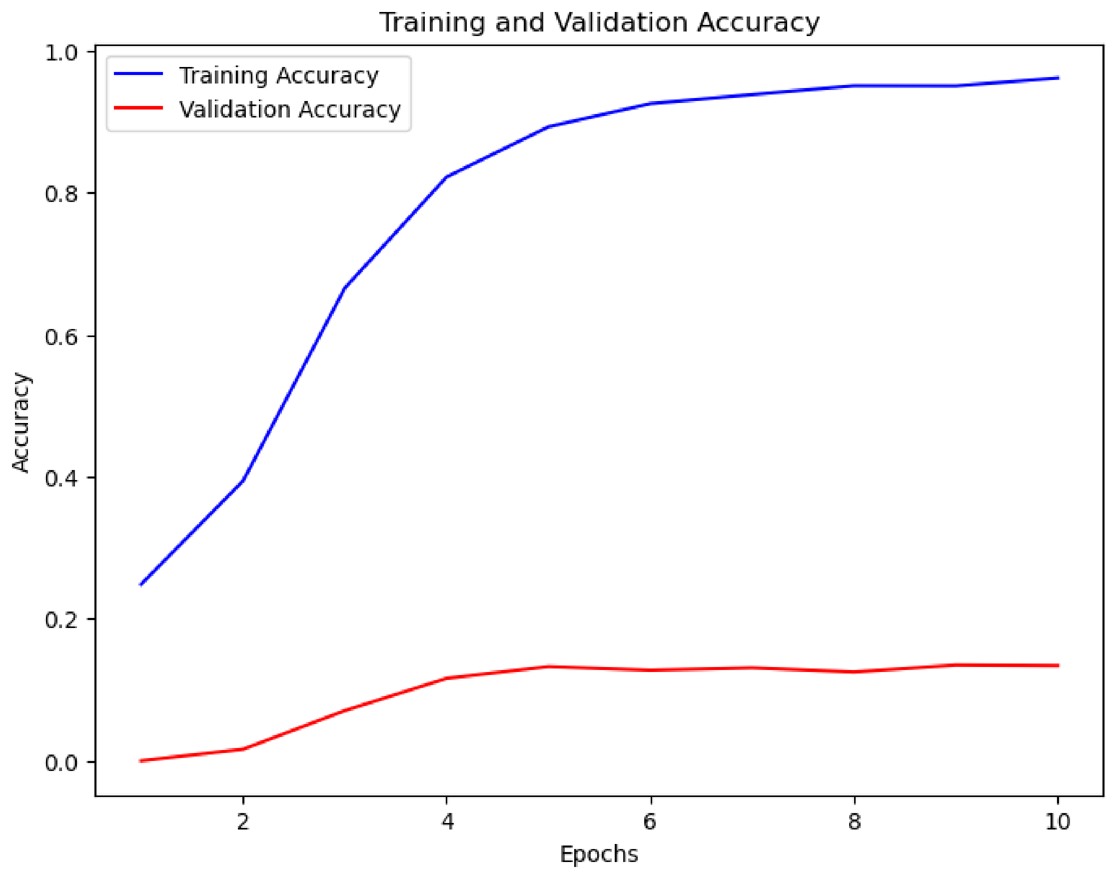
```
135/135 [==============================] - 13s 97ms/step - loss: 5.9682 -
accuracy: 0.7537
```

[66]:
```
validation_file = './Data/valid.p'
print("Load Validation Data")
with open(validation_file, mode='rb') as f:
    valid = pickle.load(f)
images_valid, labels_valid = valid['features'], valid['labels']

mask_0_to_8_valid = labels_valid <= 8
images_valid_filtered = images_valid[mask_0_to_8_valid]
labels_valid_filtered = labels_valid[mask_0_to_8_valid]

history_2 =   model.fit(images_train_filtered, labels_train_filtered, epochs=8,
 ↪validation_data=(images_valid_filtered, labels_valid_filtered))

# Define the file name for saving the model
```

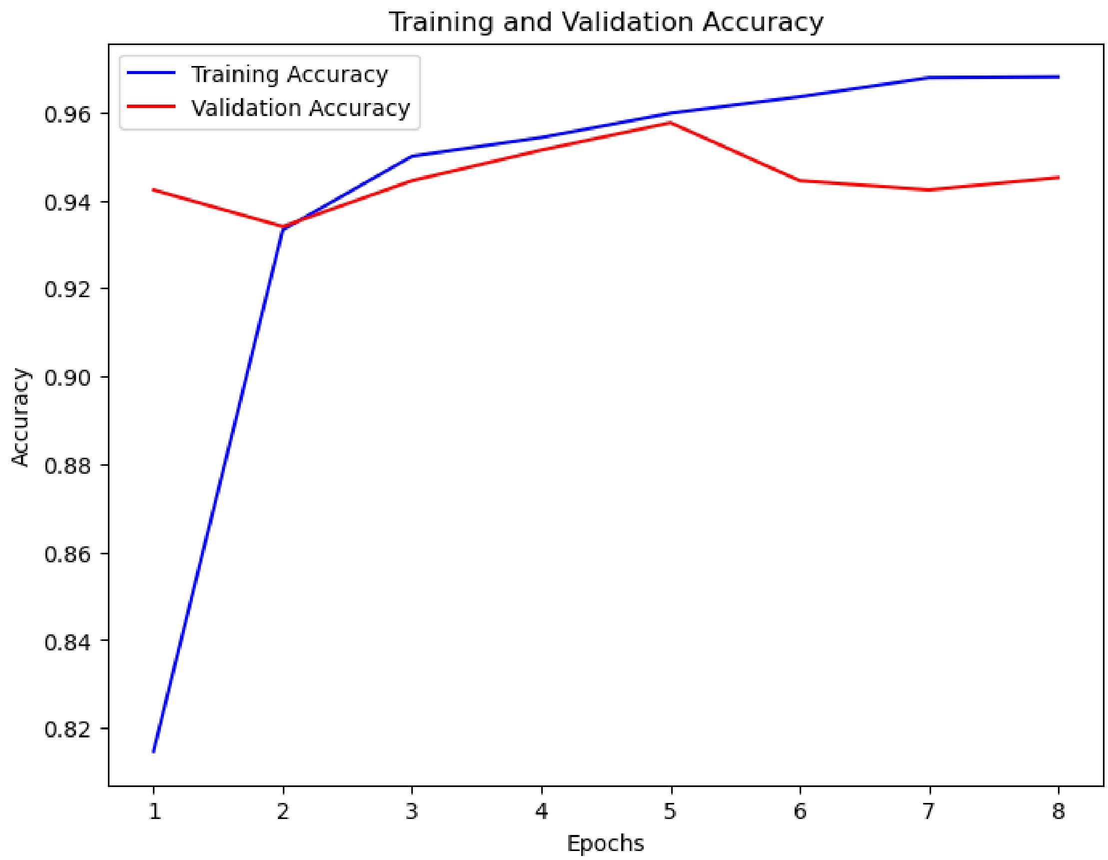
```python
model_filename = 'Convolution_Model_F1__Ex_2_2'

# Save the model to a file
model.save(model_filename)

test_file = './Data/test.p'

with open(test_file, mode='rb') as f:
    test = pickle.load(f)
images_test, labels_test = test['features'], test['labels']

mask_test_0_to_8_test = labels_test <= 8
images_test = images_test[mask_test_0_to_8_test]
labels_test = labels_test[mask_test_0_to_8_test]
```

```
Load Validation Data
Epoch 1/8
368/368 [==============================] - 127s 337ms/step - loss: 0.5899 -
accuracy: 0.8147 - val_loss: 0.1911 - val_accuracy: 0.9424
Epoch 2/8
368/368 [==============================] - 125s 339ms/step - loss: 0.2144 -
accuracy: 0.9332 - val_loss: 0.2360 - val_accuracy: 0.9340
Epoch 3/8
368/368 [==============================] - 109s 295ms/step - loss: 0.1604 -
accuracy: 0.9500 - val_loss: 0.1784 - val_accuracy: 0.9444
Epoch 4/8
368/368 [==============================] - 110s 300ms/step - loss: 0.1497 -
accuracy: 0.9543 - val_loss: 0.1543 - val_accuracy: 0.9514
Epoch 5/8
368/368 [==============================] - 111s 302ms/step - loss: 0.1385 -
accuracy: 0.9598 - val_loss: 0.1543 - val_accuracy: 0.9576
Epoch 6/8
368/368 [==============================] - 115s 313ms/step - loss: 0.1212 -
accuracy: 0.9635 - val_loss: 0.1776 - val_accuracy: 0.9444
Epoch 7/8
368/368 [==============================] - 139s 378ms/step - loss: 0.1117 -
accuracy: 0.9679 - val_loss: 0.1851 - val_accuracy: 0.9424
Epoch 8/8
368/368 [==============================] - 133s 360ms/step - loss: 0.1137 -
accuracy: 0.9680 - val_loss: 0.1648 - val_accuracy: 0.9451
INFO:tensorflow:Assets written to: Convolution_Model_F1__Ex_2_2/assets
```


```
INFO:tensorflow:Assets written to: Convolution_Model_F1__Ex_2_2/assets
```



[72]:
```
# Train the model and store the training history in a variable named "history"

# Extract accuracy values
train_accuracy = history_2.history['accuracy']
val_accuracy = history_2.history['val_accuracy']

# Plot accuracy
plt.figure(figsize=(8, 6))
epochs = range(1, len(train_accuracy) + 1)
plt.plot(epochs, train_accuracy, 'b', label='Training Accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



[70]:

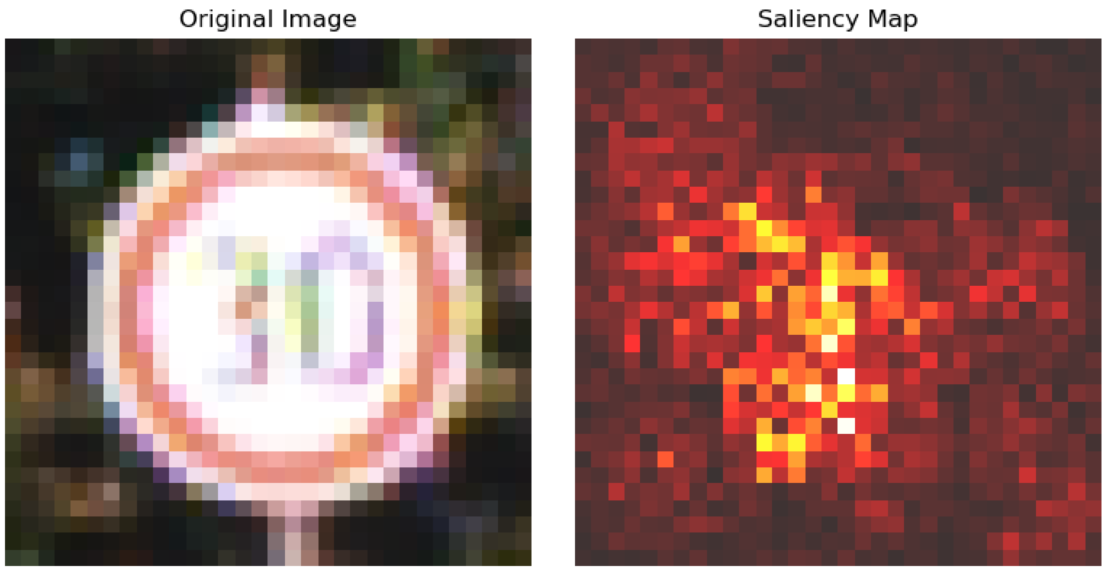
```
test_loss, test_accuracy  = model.evaluate(images_test, labels_test) # ,f1_score
```

```
135/135 [==============================] - 13s 93ms/step - loss: 0.3981 -
accuracy: 0.9125
```

[91]:

```
# Choose an index from your validation set
index = 7  # Replace with the index of the image you want to visualize

# Select an image from the validation set
selected_image = images_valid_filtered[index]

# Convert the selected image to a format accepted by the model
input_image = np.expand_dims(selected_image, axis=0)  # Add batch dimension

# Define a function to compute gradients
@tf.function
def compute_gradients(input_image):
    with tf.GradientTape() as tape:
        tape.watch(input_image)
        predictions = model(input_image)
```

```
        predicted_class = tf.argmax(predictions, axis=1)
        predicted_output = predictions[0, predicted_class[0]]  # Access the␣
↪predicted output directly
    gradients = tape.gradient(predicted_output, input_image)
    return gradients

# Compute gradients for the selected image
gradients = compute_gradients(tf.convert_to_tensor(input_image, dtype=tf.
↪float32))

# Convert gradients to a saliency map
saliency_map = tf.abs(gradients)
saliency_map = tf.reduce_max(saliency_map, axis=-1)
saliency_map = saliency_map.numpy()[0]

# Normalize the saliency map
saliency_map = (saliency_map - saliency_map.min()) / (saliency_map.max() -␣
↪saliency_map.min())

# Visualize the saliency map overlaid on the original image
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.imshow(selected_image)
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(saliency_map, cmap='hot', alpha=0.8)
plt.title('Saliency Map')
plt.axis('off')

plt.tight_layout()
plt.show()
```



[93]:

```
  Cell In[93], line 1
    half the images, double the images (for overfitting)
         ^
SyntaxError: invalid syntax
```