Project: 'Deep Learning: Image Classification with CNN'

Report

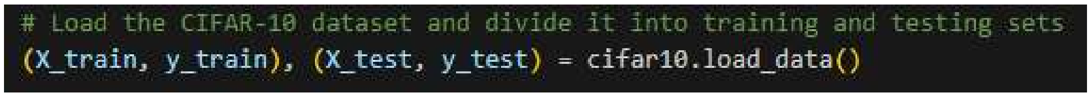
Jaime

December, 2024

1. Loading the data set ‘CIFAR-10’

We tried to different options:

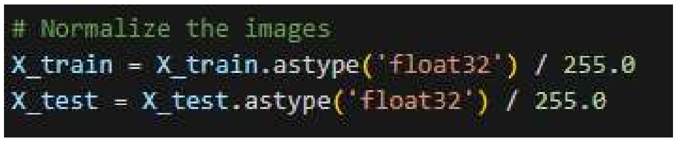
- Without any download. I imported it directly into the jupyter notebook using ‘tensorflow.keras.datasets’, which allowed me to separate both training and testing sets in a single code line.



```python
# Load the CIFAR-10 dataset and divide it into training and testing sets
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

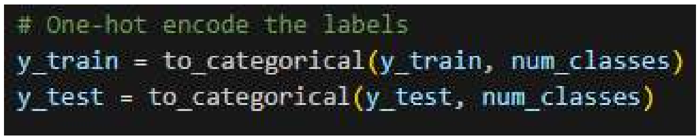
2. Preprocessing Steps

- Normalization: Images were converted into floats and scaled to have pixel values between 0 and 1, which helps the model converge faster.



```python
# Normalize the images
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
```

- One-hot encoding: Labels were converted to categorical format to suit the categorical cross-entropy loss function.



```python
# One-hot encode the labels
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
```

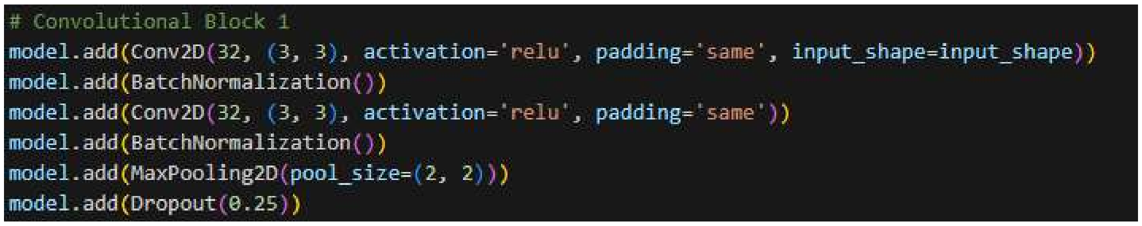
3. Model building and hyperparameters

The model has various **4 convolutional blocks** with different filters, using ReLu as the activation function and padding set on ‘same’, so we don’t lose any dimensions.

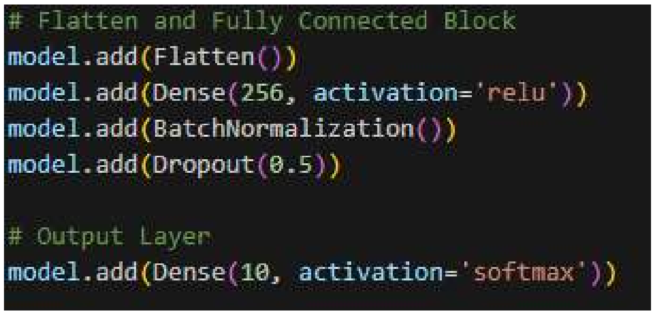
In between each Conv2D layer I added a **batch normalization** that helps to stabilize and faster the training.

Each block of Conv2D is separated by **Max-pooling** to reduce spatial dimensions and a **Dropout (**that increases its size rate in each block) to help reduce the overfitting.

at the end the model was **flattened** and added another Dense layer to fully connect the layers before the ‘softmax’ function that ensures the output



```
# Convolutional Block 1
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=input_shape))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```



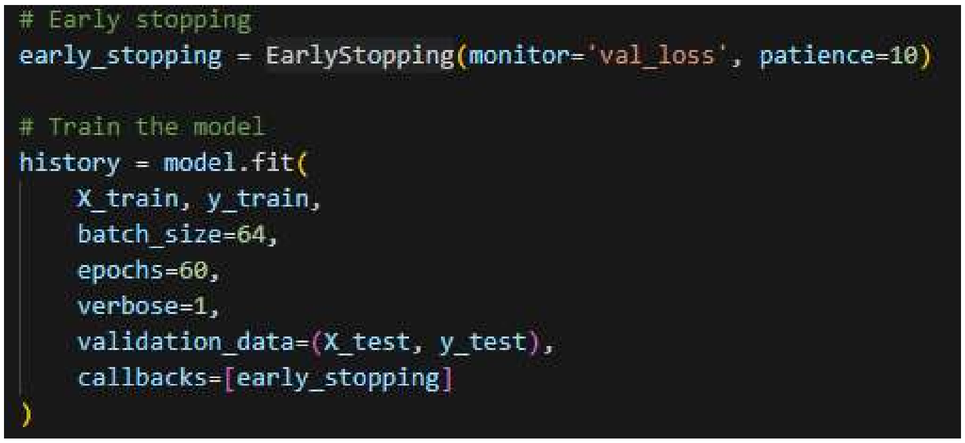
```
# Flatten and Fully Connected Block
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Output Layer
model.add(Dense(10, activation='softmax'))
```

4. Training Process

While **compiling** the model, I used the Adam optimizer and chose the accuracy metric to track the performance of our model. As the loss function I used ‘categorical_crossentropy’.

The model was trained with a sample size of 50,000 images. I set the number of epochs to 60, the batch size to 64 and the sample size to 10,000. To prevent overfitting and unnecessary compute time I used EarlyStopping.

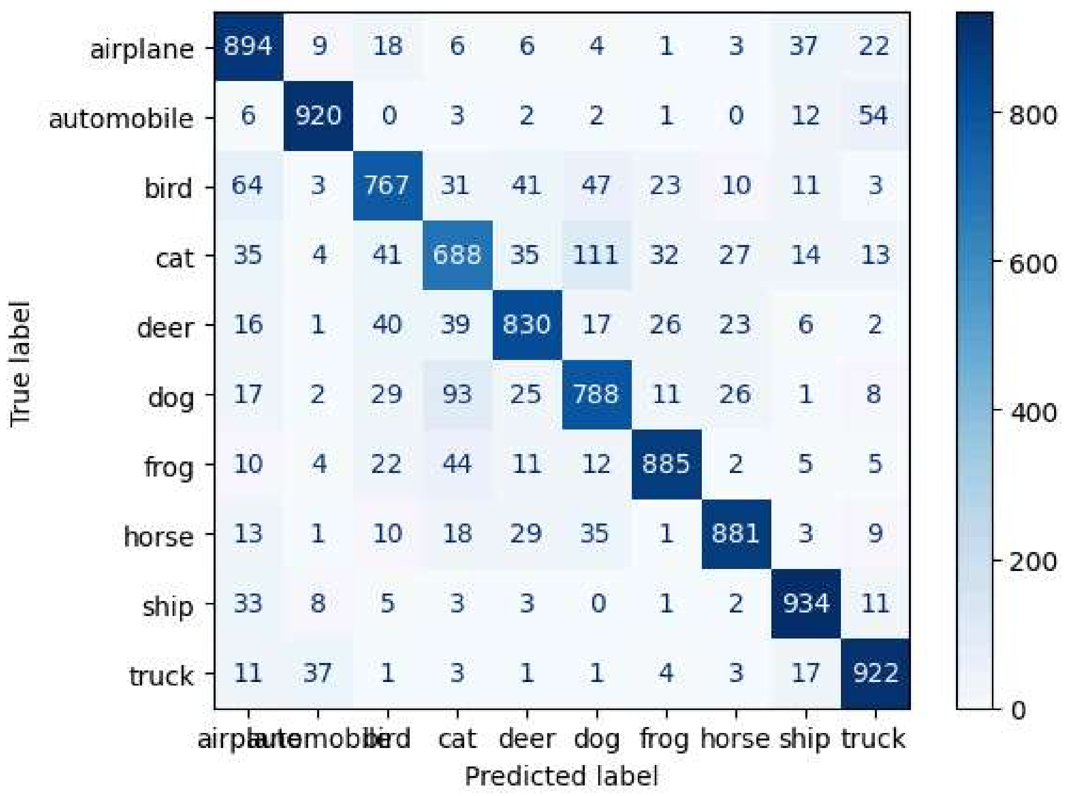


```
# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10)

# Train the model
history = model.fit(
    X_train, y_train,
    batch_size=64,
    epochs=60,
    verbose=1,
    validation_data=(X_test, y_test),
    callbacks=[early_stopping]
)
```

5. Model performance (evaluation metrics)

Training the model got it to a training accuracy of **96.82%** in just 11 epoch where it exited thanks to EarlyStopping. After computing the accuracy score on the test data, the CNN model achieved a test accuracy of **85%** and a test loss of 0.64% on unseen data.



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	894	9	18	6	6	4	1	3	37	22
automobile	6	920	0	3	2	2	1	0	12	54
bird	64	3	767	31	41	47	23	10	11	3
cat	35	4	41	688	35	111	32	27	14	13
deer	16	1	40	39	830	17	26	23	6	2
dog	17	2	29	93	25	788	11	26	1	8
frog	10	4	22	44	11	12	885	2	5	5
horse	13	1	10	18	29	35	1	881	3	9
ship	33	8	5	3	3	0	1	2	934	11
truck	11	37	1	3	1	1	4	3	17	922

Confusion matrix revealed:

- Strengths in categories like airplanes, ships and trucks.
- Confusion between classes like cats and dogs or trucks and automobiles, which visually are very similar.

6. Transfer Learning

I chose VGG16 because it’s widely used architecture for image classification tasks compared to models like Inception or ImagNet, making it easier to fine-tune for a dataset like CIFAR-10, which contains smaller 32x32 images..

VGG16

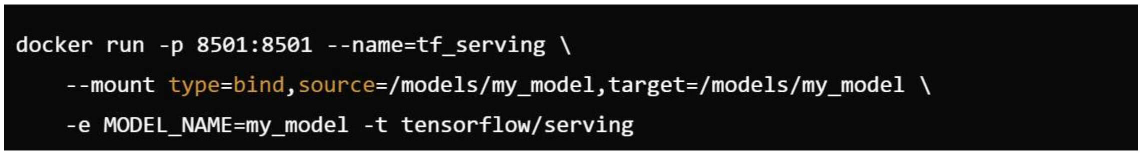
- Test loss: 1.716
- Test accuracy: 0.652

6. Save and load

I saves the model using model.export('saved_model/my_model/1/') which is the recommended mode if you're planning to use Tensorflow Serving

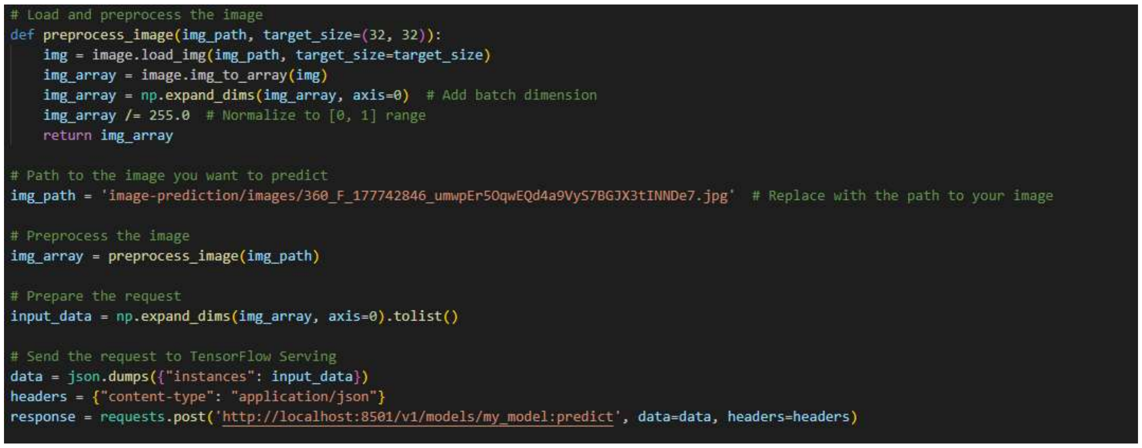
7. Tensor Flow Serving

I created a tensorflow serving docker container using my model and the tensorflow serving image provided by google using this command:



```
docker run -p 8501:8501 --name=tf_serving \
    --mount type=bind,source=/models/my_model,target=/models/my_model \
    -e MODEL_NAME=my_model -t tensorflow/serving
```

then I ran the container in docker and created a python request code snipped to test the API



```
# Load and preprocess the image
def preprocess_image(img_path, target_size=(32, 32)):
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    img_array /= 255.0  # Normalize to [0, 1] range
    return img_array

# Path to the image you want to predict
img_path = 'image-prediction/images/360_F_177742846_umwpEr5OqwEQd4a9VyS7BGJX3tINNDe7.jpg'  # Replace with the path to your image

# Preprocess the image
img_array = preprocess_image(img_path)

# Prepare the request
input_data = np.expand_dims(img_array, axis=0).tolist()

# Send the request to TensorFlow Serving
data = json.dumps({"instances": input_data})
headers = {"content-type": "application/json"}
response = requests.post('http://localhost:8501/v1/models/my_model:predict', data=data, headers=headers)
```

8. Hosting the model in google cloud

This is an overview of the process to host the model in GCP

A. **Upload the model** to Google Cloud Storage (GCS).
B. **Use Vertex AI** to deploy the model and create an endpoint for serving predictions.
C. **Send requests to the endpoint** from your website or client application to make predictions.

9. Flask webapp