Project I | Deep Learning: Image Classification with CNN

1. Introduction

1.1 Project Overview

The goal of the project is to build a Convolutional Neural Network (CNN) model to classify images from the CIFAR-10 dataset into 10 predefined categories/classes.

1.2 Data Set

The dataset used for this project is the CIFAR-10 dataset, which consists of 60,000 32x32 color images from 10 classes, with 6,000 images per class.

1.3 Tools and Technologies

- Python
- Google.Colab, Jupyter Notebook, Tensorflow Serving, Docker Hub
- Docker, PuTTY, WinSCP
- Libraries: tensorflow, NumPy, Scikit-Learn, Matplotlib, Seaborn

2. Data Processing

After importing the CIFAR-10 dataset from Keras library and defining class names, I split the data set into training, validation, and test sets, where 20% were allocated for validation.

I started data processing by visualizing sample images to verify their integrity and correct labeling. Next, I normalized the image data by scaling and augmenting data using such techniques as rotation, shifting, shearing, zooming, and flipping to enhance the model's ability to generalize from varied input data.

I proceeded to format the images and labels and prepared data generators to feed them into the model in batches. Finally, labels are one-hot encoded to transform them into a suitable format for classification tasks. This comprehensive preprocessing ensures the dataset is ready for effective model training.

3. Model Development and Results

I tested various model architectures with different parameters starting with a basic model considered in one of the labs:

Code Snippet:

```
#Define a cnn model
def build_cnn_model():
    cnn_model = tf.keras.Sequential([
        # First convolutional layer
        tf.keras.layers.Conv2D(16, (5, 5), strides=(1, 1),
        activation='relu', input_shape=(32, 32, 3)),

        # First max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)),

        # Second convolutional layer
        tf.keras.layers.Conv2D(8, (5, 5), strides=(1, 1),
        activation='relu'),

        # Second max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),

        # Output layer
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    return cnn_model
```

The model showed 0.553 accuracy on validation set and 0.547 accuracy on test set. The model was compiled with Adam optimizer and trained over 5 epochs.

The following steps had marginal effect on the accuracy of the model (see file 'Project_Week_3_choice_of_cnn' for more details):

- Increasing the number of epochs to 100,
- Setting learning rate to 0.001,
- Setting learning rate to 0.0001,
- Changing optimizer to RMSprop,
- Adding another convolutional layer,
- Adding another dense layer.

The metrics of the models with higher accuracy are summarized in Table 1, including models with transfer learning without fine tuning (see file 'Project_week_3_Knowledge_transfer') and with fine tuning (Project_week_3_Transfer_Learning_fine_tuned_model').

Table 1. Metrics of models with higher accuracy.

	Doubled number of filters and neurons in the model with three convolutional layers	Added 3d MaxPool layer and batch normalizations after every layer	Benchmark from the web	InceptionV3	VGG16	VGG16 fine tuned
Validation accuracy	0.632	0.666	0.818	0.643	0.697	0.591
Test accuracy	0.636	0.686	0.817	0.639	0.692	0.524
Precision	0.635	0.688	0.819	0.645	0.703	0.580
Recall	0.636	0.686	0.817	0.639	0.693	0.524
F1	0.633	0.685	0.815	0.638	0.691	0.532

Source: own elaboration.

The architecture of the benchmark model (with small alterations) from the web is shown below.

Code Snippet:

```
#Define a cnn model
def build_cnn_model7():
    cnn_model7 = tf.keras.Sequential([
        tf.keras.layers.Conv2D(32, (3, 3), strides=(1, 1),
        activation='relu', input_shape=(32, 32, 3)),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(32, (3, 3), strides=(1, 1),
        activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)),
        tf.keras.layers.Dropout(0.3),

        tf.keras.layers.Conv2D(64, (3, 3), strides=(1, 1),
        activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(64, (3, 3), strides=(1, 1),
        activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)),
        tf.keras.layers.Dropout(0.5),
```

```
        tf.keras.layers.Conv2D(128, (3, 3), strides=(1, 1),
        activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(128, (3, 3), strides=(1, 1),
        activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPool2D(pool_size=(1, 1), strides=(1, 1)),
        tf.keras.layers.Dropout(0.5),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(0.5),
        # Output layer
        tf.keras.layers.Dense(10, activation='softmax')

        ])

    return cnn_model7
```

The architecture demonstrated the highest performance on all metrics and was selected for deployment.

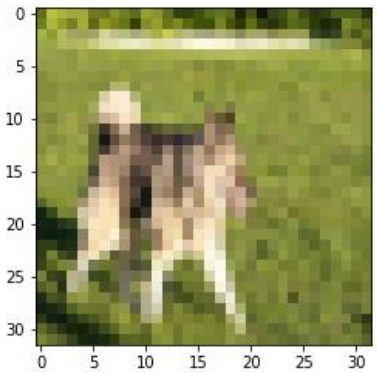
4. Deployment

4.1 Local Deployment

I deployed the best performing image classification model locally using Flask and TensorFlow Serving. See directory ‘my_model/3’; the model is reached under: http://127.0.0.1:5000.

5 images were used to test the predictions of the model.

The models correctly classified the following images:



The model incorrectly classified the following image as ‘dog’:

4.2 Attempt at Deploying on Amazon Web Services (AWS)

I took the following steps to deploy the model on AWS:

- Containerized my model into a Docker container;
- Set up account on AWS and launched and configured an EC2 instance;
- Connected to EC2 instance using PuTTY;
- Loaded the files to EC2 instance using Win SCP;
- Installed Docker on EC2;
- Encountered error when starting a container based on image on EC2 instance:

  docker: Error response from daemon: failed to create task for container: failed to create shim task: OCI runtime create failed: runc create failed: unable to start container process: exec: "tensorflow_model_server": executable file not found in $PATH: unknown.

A solution proposed on Stackflow suggests creating a separate Dockerfile that only runs the TensorFlow server, which I could not complete in time.

(see: https://stackoverflow.com/questions/70355410/not-able-to-run-linux-command-in-background-from-dockerfile).

The files for this attempt can be found under ‘my_model/4’.

5. Conclusion and Future Work

5.1 Conclusion

The project successfully developed and deployed an image classification model, where the architecture with multiple alternating convolutional, max-pooling, batch normalization and spatial reduction layers was selected.

5.2 Key Takeaways

I would like to mention some key learnings, where the project helped me identify some areas of improvement:

- The importance of monitoring data input and output shape.
- Caution in situations of exceptionally good model performance.

5.3 Future Work

Next steps include improving performance of the model and deploying the model on AWS or similar platform.