Report on Deep Learning Image Classification Project

1. Introduction

Project Overview:

The objective of this project is to develop a Convolutional Neural Network (CNN) model to classify images from the CIFAR-10 dataset into one of 10 categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

Dataset:

The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. The dataset is divided into 50,000 training images and 10,000 test images.

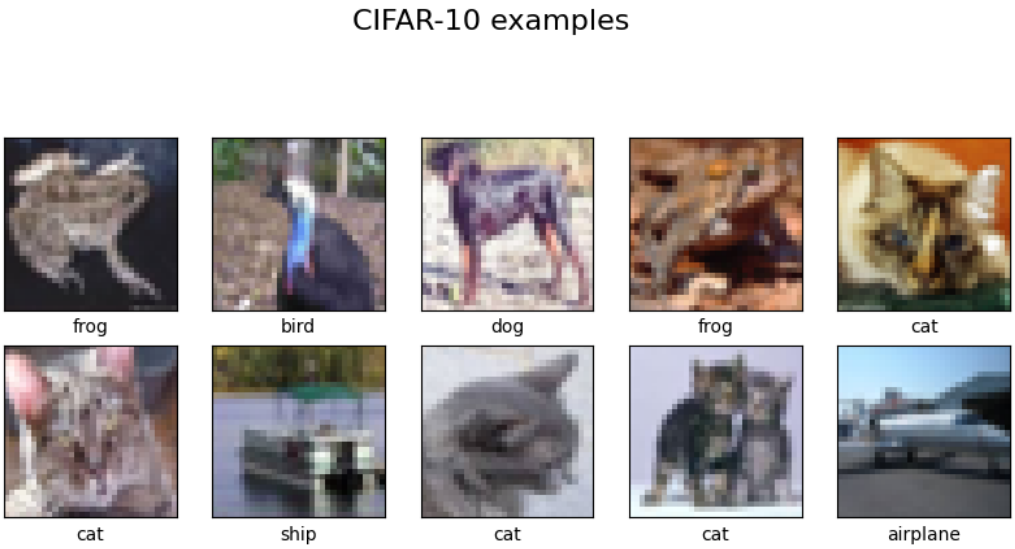
Tools and Technologies:

1. Colab for development environment
2. Python as development language
3. TensorFlow and Keras for model development
4. NumPy and Pandas for data manipulation
5. Matplotlib and Seaborn for data visualization

2. Data Exploration and Preprocessing

Data Exploration:

Initially, the dataset was explored to understand its structure and characteristics. This included loading the data, visualizing sample images, and analyzing the distribution of classes.



Preprocessing Steps:

Resizing: In this project, the images are not resized because the chosen CNN architecture accept the original size of CIFAR-10 images, which is 32x32 pixels. Resizing images can sometimes lead to loss of important details and distortions, which may negatively impact the model's performance.

Normalization: Pixel values of the images were normalized to the range [0, 1] to facilitate faster convergence during training.

```
# Normalization
train_images = train_images / 255.0
test_images = test_images / 255.0
val_images = val_images / 255.0
```

Data Augmentation: Techniques such as rotation, width and height shifts, horizontal flips, and zoom were applied to increase the diversity of the training data and reduce overfitting.

```
# Augmentation
datagen = ImageDataGenerator(
    rotation_range = 40,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    horizontal_flip = True,
    zoom_range = 0.2,
    fill_mode = 'nearest'
)
datagen.fit(train_images)
```

3. Model Development

At first the accuracy results where very bad, like 0.3 or 0.4. Several approaches where tested, such as using more layers, different optimizers and learning rates.

Chosen CNN Architecture:

The chosen architecture consists of several convolutional layers followed by batch normalization and dropout layers to prevent overfitting. This architecture effectively captures spatial hierarchies in the images.

Two functions were created to make these testing simpler and avoid code repetition.

```python
def create_cnn_layers(input_tensor):
    """
    Creates the convolutional layers for a CNN model.

    Args:
        input_tensor (Tensor): Input tensor to the CNN layers.

    Returns:
        Tensor: Output tensor after applying the convolutional, batch normalization,
                max pooling, flatten, dense, and dropout layers.

    This function builds a series of convolutional, batch normalization, and max pooling layers to extract features
    from the input tensor.
    """
    # First Convolutional layer with 32 filters to receive the input and capture low-level features like edges, textures, etc.
    x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_tensor)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D((2, 2))(x)
    # Second  Convolutional layer with 64 filters to capture, builds upon the features detected by first layer, complex features.
    x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D((2, 2))(x)
    # Third  Convolutional layer with 128 filters to capture, builds upon the features detected by second layer, more complex features.
    x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D((2, 2))(x)
    # Fourth  Convolutional layer with 128 filters to capture, builds upon the features detected by third layer, more complex features.
    x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = layers.BatchNormalization()(x)
    # Flatten layer to convert 3D output to 1D feature vector.
    x = layers.Flatten()(x)
    # Dense layer with 128 units and ReLU activation.
    x = layers.Dense(128, activation='relu')(x)
    # Dropout layer with 0.5 dropout rate for regularization.
    x = layers.Dropout(0.5)(x)
    # Final Dense layer with 10 units (classes) and softmax activation for classification.
    output_tensor = layers.Dense(10, activation='softmax')(x)
    return output_tensor

def create_cnn_model():
    """
    Creates and returns a compiled CNN model for CIFAR-10 classification.

    This function defines the input shape for the model, constructs the convolutional layers by
    calling the create_cnn_layers function, and then creates the Model object.

    Returns:
        model (Model): A compiled Keras Model object.
    """
    input_tensor = layers.Input(shape=(32, 32, 3))
    output_tensor = create_cnn_layers(input_tensor)
    model = models.Model(inputs=input_tensor, outputs=output_tensor)
    return model
```

Compilling the Model:

```python
# Compile the model
model_cn.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])
```

Optimizer: Adam was chosen for its adaptability, efficiency, and robustness, making it suitable for training complex CNN architectures on datasets like CIFAR-10.

Loss: Categorical Cross-Entropy is well-suited for multi-class classification tasks like CIFAR-10, where each image belongs to one of ten distinct classes.

Metrics: Accuracy is used to easily quantify and compare the model's performance, making it a preferred choice for reporting results.

Training the Model:

```
# Train the model
history = model_cn.fit(
    datagen.flow(train_images, to_categorical(train_labels), batch_size=32),
    epochs = 100,
    validation_data=(val_images, to_categorical(val_labels)),
    callbacks=[early_stopping, lr_scheduler]
)
```

Learning Rate: The model was trained with a learning rate of 0.001.

Batch Size: A batch size of 32 was used for training.

Epochs: The model was trained for up to 100 epochs with early stopping and learning rate scheduling to prevent overfitting.

```
# Define early stopping function
early_stopping = callbacks.EarlyStopping(
    monitor='val_loss',
    patience=10,
    verbose=1,
    restore_best_weights=True
)

# Define lerning rate scheduler function
lr_scheduler = callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=5,
    verbose=1
)
```

Model Performance:

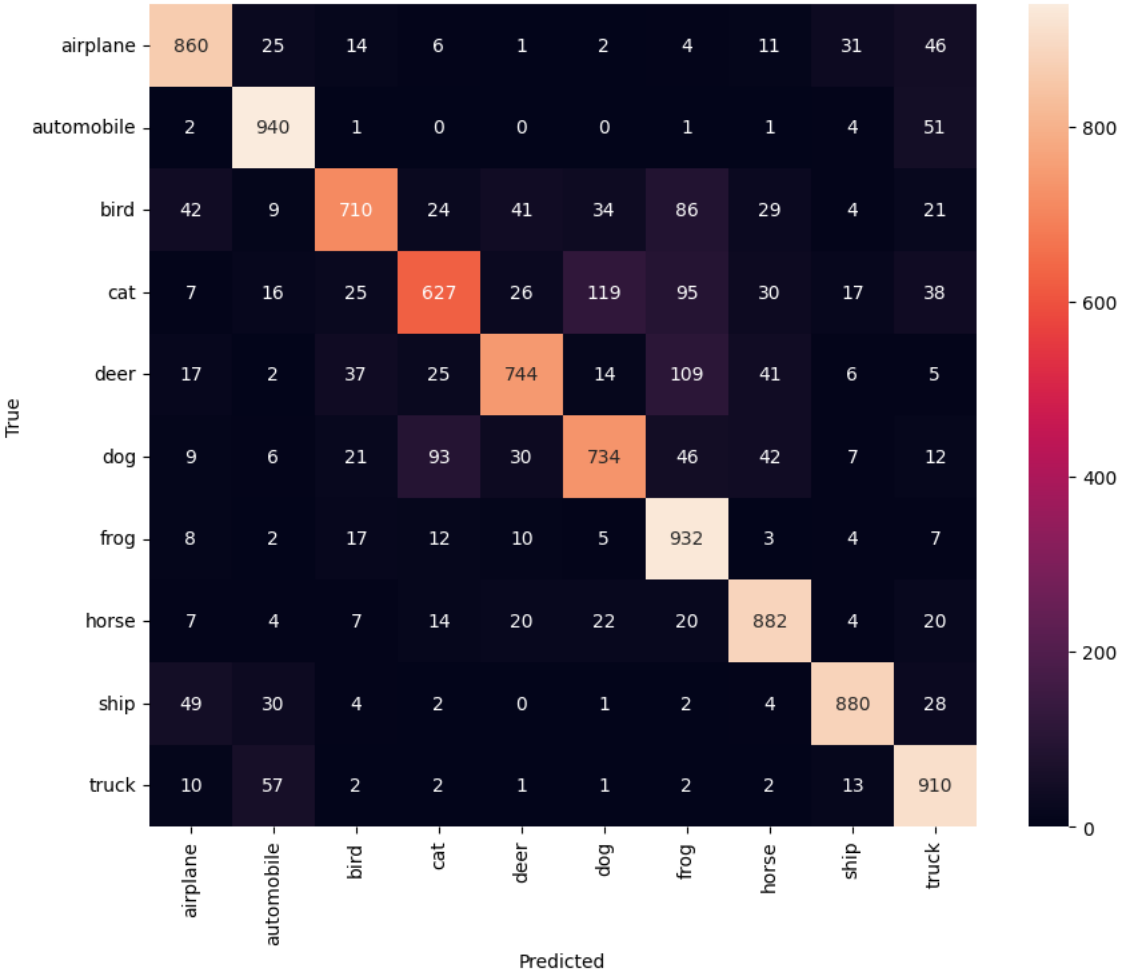
The model was evaluated on the test set achieving a test accuracy of **0.81**.

```
313/313 - 1s - loss: 0.5892 - accuracy: 0.8055 - 734ms/epoch - 2ms/step
Test accuracy: 0.81
313/313 [==============================] - 1s 2ms/step
Accuracy: 0.81
```

```
Classification Report:
              precision    recall  f1-score   support

    airplane       0.85      0.86      0.86      1000
  automobile       0.86      0.94      0.90      1000
        bird       0.85      0.71      0.77      1000
         cat       0.78      0.63      0.69      1000
        deer       0.85      0.74      0.79      1000
         dog       0.79      0.73      0.76      1000
        frog       0.72      0.93      0.81      1000
       horse       0.84      0.88      0.86      1000
        ship       0.91      0.88      0.89      1000
       truck       0.80      0.91      0.85      1000

    accuracy                           0.82     10000
   macro avg       0.82      0.82      0.82     10000
weighted avg       0.82      0.82      0.82     10000
```



4. Transfer Learning

The InceptionV3, ResNet50 and VGG16 models were chosen for transfer learning approch.

Comparing Models:

Were created the functions compile_and_train and create_model, to fairly compare how each model peforms.

```
def compile_and_train(model, model_name):
    """
    Compiles and trains the given model, and evaluates its performance.

    Args:
        model (tf.keras.Model): The model to be trained and evaluated.
        model_name (str): The name of the model (used for display purposes).

    This function compiles the model with the Adam optimizer and categorical crossentropy loss.
    It adjusts the image sizes for InceptionV3 as it requires larger input dimensions.
    The model is trained using data augmentation, and the training history is recorded.
    Finally, the model's performance is evaluated on the test set, and metrics such as accuracy,
    precision, recall, and F1-score are printed.
    """

    # Compile the model
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    # Adjust image sizes for InceptionV3
    if model_name == "InceptionV3":
        _train_images = tf.image.resize(train_images, (75, 75))
        _val_images = tf.image.resize(val_images, (75, 75))
        _test_images = tf.image.resize(test_images, (75, 75))
    else:
        _train_images = train_images
        _val_images = val_images
        _test_images = test_images

    # Train the model
    history = model.fit(
        datagen.flow(_train_images, to_categorical(train_labels), batch_size=32),
        epochs=20,
        validation_data=(_val_images, to_categorical(val_labels)),
        callbacks=[early_stopping, lr_scheduler]
    )

    # Evaluate the model
    test_predictions = model.predict(_test_images).argmax(axis=1)
    test_labels_flat = test_labels.flatten()

    accuracy = accuracy_score(test_labels_flat, test_predictions)
    print(f'{model_name} Model Accuracy: {accuracy:.2f}')

    precision = precision_score(test_labels_flat, test_predictions, average=None)
    recall = recall_score(test_labels_flat, test_predictions, average=None)
    f1 = f1_score(test_labels_flat, test_predictions, average=None)

    print(f'Precision: {precision}')
    print(f'Recall: {recall}')
    print(f'F1-score: {f1}')

    report = classification_report(test_labels_flat, test_predictions, target_names=class_names)
    print(f'Classification Report for {model_name}:\n', report)
```

```python
def create_model(base_model):
    """
    Creates a model by adding custom top layers to a base pre-trained model.

    Args:
        base_model (tf.keras.Model): The base pre-trained model (e.g., VGG16, InceptionV3, ResNet50).

    Returns:
        tf.keras.Model: The combined model with the base model and custom top layers.

    This function freezes the layers of the base model to prevent them from being trained further.
    """

    # Freeze the base model layers
    for layer in base_model.layers:
        layer.trainable = False

    # Add custom top layers
    x = base_model.output
    x = layers.Flatten()(x)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    predictions = layers.Dense(10, activation='softmax')(x)

    # Create the model
    model = models.Model(inputs=base_model.input, outputs=predictions)
    return model
```

Although the parametrers are the same that are used in the custom-made CNN model, the layers and the epoch size differ from the original model, witch isn't as relevant to finding with greater accuracy. Note that to use pre-treined model InceptionV3 it was necessary to modify the shape of input layer and resize the images.

```python
# Inputs tensor
input_tensor = layers.Input(shape=(32, 32, 3))
input_tensor_inception = layers.Input(shape=(75, 75, 3))

# VGG16 model
vgg16_base = VGG16(weights='imagenet', include_top=False, input_tensor=input_tensor)
vgg16_model = create_model(vgg16_base)

# InceptionV3 model
inception_base = InceptionV3(weights='imagenet', include_top=False, input_tensor=input_tensor_inception)
inception_model = create_model(inception_base)

# ResNet50 model
resnet_base = ResNet50(weights='imagenet', include_top=False, input_tensor=input_tensor)
resnet_model = create_model(resnet_base)

# Train and evaluate each model
compile_and_train(vgg16_model, "VGG16")
compile_and_train(inception_model, "InceptionV3")
compile_and_train(resnet_model, "ResNet50")
```

Results:

Model	Accuracy
VGG16	0.55
InceptionV3	0.60
ResNet50	0.10

InceptionV3 achieved better accuracy, but to use it with the CIFAR-10 dataset requires resizing images from 32x32 to 75x75 and this process can introduce artifacts and blur the image, potentially losing fine details that can affect the model's ability to learn precise features.

VGG16 was chosen because it has the second best accuracy and can be used with the CIFAR-10 dataset without the need to resize images.

Combined Model:

```
# When I was using MaxPoolin2D layers, the features maps were very small and I was getting error
def create_cnn_layers_without_maxpooling2D(input_tensor):
  """
  Creates CNN layers without MaxPooling2D to prevent feature maps from becoming too small.

  Args:
      input_tensor (Tensor): Input tensor to the CNN layers.

  Returns:
      Tensor: Output tensor after applying convolutional, batch normalization, flatten, dense, and dropout layers.

  This function builds a series of convolutional and batch normalization layers to extract features
  from the input tensor without using MaxPooling2D layers.
  """
  x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_tensor)
  x = layers.BatchNormalization()(x)
  x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
  x = layers.BatchNormalization()(x)
  x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
  x = layers.BatchNormalization()(x)
  x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
  x = layers.BatchNormalization()(x)
  x = layers.Flatten()(x)
  x = layers.Dense(128, activation='relu')(x)
  x = layers.Dropout(0.5)(x)
  output_tensor = layers.Dense(10, activation='softmax')(x)
  return output_tensor


# Load VGG16 model without the top layers
vgg16_base = VGG16(weights='imagenet', include_top=False, input_tensor=input_tensor)
# Freeze the layers of the base model to prevent them from being trained
for layer in inception_base.layers:
    layer.trainable = False
# Create custom layers
vgg16_output = vgg16_base.output
cnn_output = create_cnn_layers_without_maxpooling2D(vgg16_output)
# Create the combined model
combined_model = models.Model(inputs=vgg16_base.input, outputs=cnn_output)
```

When the combined model was built it was necessary to change the architecture of the CNN model because the use of MaxPooling2D layers caused an error causing the feature maps to be too small.

```
# Compile the combined model
combined_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                       loss='categorical_crossentropy',
                       metrics=['accuracy'])

# Train the combined model
history_combined = combined_model.fit(
    datagen.flow(train_images, to_categorical(train_labels), batch_size=32),
    epochs=100,
    validation_data=(val_images, to_categorical((val_labels))),
    callbacks=[early_stopping, lr_scheduler]
)
```

The parameters are the same as for the initial training of the CNN model. And the accuracy achieved was 0.56.

5. Fine Tunning

The first CNN model achieved better results of accuracy (0.82) that the others, so it was trained with a fine tunning.

```
# Create the final CNN model
final_model = create_cnn_model()
# Compile the model with a lower learning rate for fine-tuning
final_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
                       loss='categorical_crossentropy',
                       metrics=['accuracy'])
# Train the model
history_finetune = final_model.fit(
    datagen.flow(train_images, to_categorical(train_labels), batch_size=32),
    epochs=100,
    validation_data=(val_images, to_categorical(val_labels)),
    callbacks=[early_stopping, lr_scheduler]
)
```

The learning rate was reduced to 0.0001 and after the train the accuracy achieved was 0.76.

6. Conclusion and Future Work

Conclusions:

- The model with the best performance is the custom-made CNN model wich achieved an accuracy rate of 81%.
- Reducing the learning rate to improve model's accuracy may not be effective if you keep the same epoch value.

- Using early stopping and learning rate scheduler callback functions improves the training process and avoids spending too much time and computational resources.
- The transfer learning approach was not achieve good results with CIFAR-10 dataset since they're optimized to different image sizes, and the dataset of each work influences the techniques used in the work.

Future Work:

The next step will be to deply the first CN model (0.81 accuracy) as a web service and provide a web page to uploading images, making it accessible for real-time predictions.