DEEP LEARNING

TECHNICAL REPORT

Group 2 – 12:00PM-1:30PM

Paul Emmanuel Corsino

David Emmanuel Lacsao

Klinnsonveins Yee

March 7, 2024

Background:

The CIFAR100 dataset is a widely used benchmark in the field of computer vision and deep learning. It consists of 60,000 32x32 color images in 100 classes, with each class containing 500 training images and 100 testing images. The dataset is divided into 20 superclasses, each containing 5 fine-grained classes. This dataset serves as a challenging task for image classification due to its small image size and diverse range of classes.

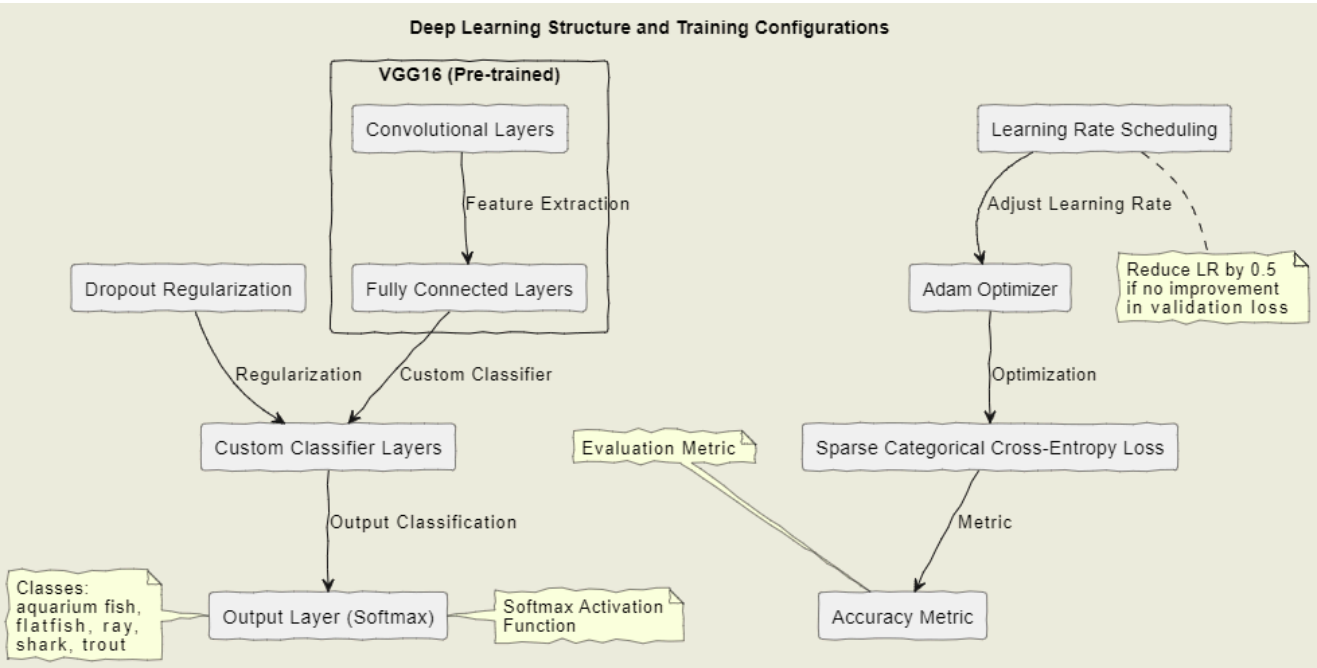
Objective:

Our objective is to build a deep-learning model for image classification using the CIFAR100 dataset. Specifically, our focus will be on classifying images belonging to the "Fish" superclass, which includes fine-grained classes such as aquarium fish, flatfish, ray, shark, and trout. Through leveraging deep learning techniques, we aim to develop a model capable of accurately categorizing images into their respective fine-grained classes within the Fish superclass.

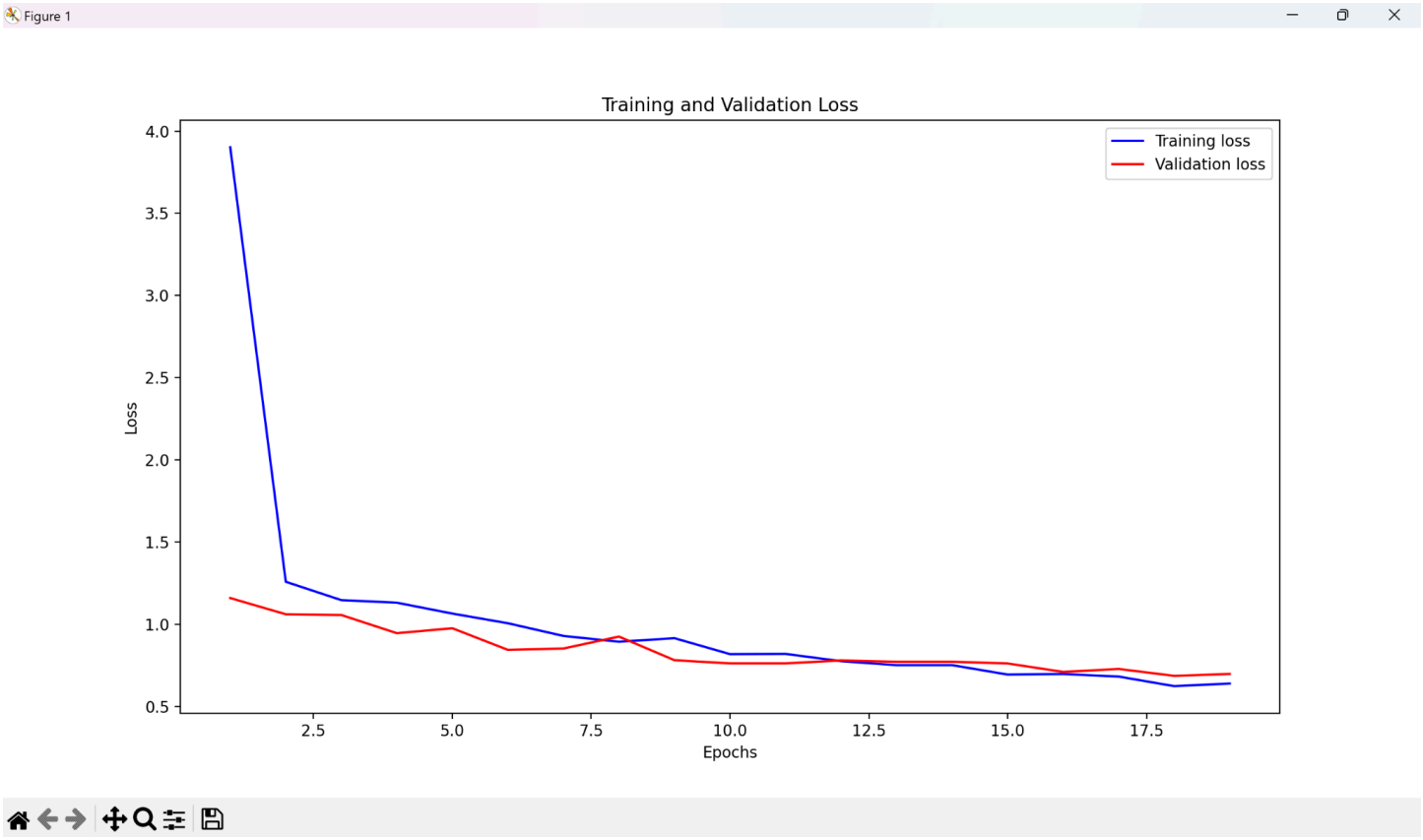
Deep Learning Structure and Training Configurations:

In our approach to deep learning, we've utilized transfer learning with the VGG16 model, incorporating it into a convolutional neural network (CNN) architecture. Freezing the pre-trained layers and adding custom classifier layers, we aimed to leverage the VGG16's feature extraction capabilities while adapting the model to classify images into five fine classes: aquarium fish, flatfish, ray, shark, and trout. We employed dropout regularization to mitigate overfitting in the fully connected layer, followed by a softmax activation function for multi-class classification.

We configured our model for training using the Adam optimizer and sparse categorical cross-entropy loss, with accuracy as the evaluation metric. Additionally, we implemented a learning rate scheduling strategy, reducing the learning rate by a factor of 0.5 after three epochs of no improvement in validation loss. Despite these measures, the model's accuracy remains inconsistent, indicating potential overfitting.



Graph of Training Loss and Validation Loss:



The graph illustrates the training loss (in blue) and the validation loss (in red) across epochs. Initially, both losses decrease as the model learns from the data. However, after a certain point (around epoch 12), while the training loss continues to decrease, the validation loss appears to stagnate and even slightly increases. This divergence indicates that the model might be overfitting to the training data, as it performs well on training data but fails to generalize to unseen validation data. In conclusion, based on the observed behavior of the loss curves, it's likely that the model starts to overfit after approximately the 12th epoch.

Challenges and Solution:

In our project about image classification, we have issues in achieving satisfactory accuracy levels. Even with our best efforts, our model's accuracy fluctuates between 68% to 76%, this could

potentially lead to a case of overfitting as both training and validation accuracies rise together. The running of our program takes a long time due to a large dataset and also a large amount of epochs. To address this, we plan to implement techniques like dropout regularization to prevent overfitting and reduce the complexity of our model. Additionally, we'll explore strategies to optimize hyperparameters, such as learning rate scheduling, to improve convergence speed and overall performance. Through experimentation and adaptation, we aim to overcome these challenges and enhance the effectiveness of our image classification model.

Conclusion:

This exercise demonstrates how AI, even in a relatively simple context like image classification using the CIFAR100 dataset, can provide valuable insights and solutions applicable across a wide range of domains, including business, consumer markets, and industries.

To begin, AI-driven image classification models can help businesses automate tasks like product categorization and quality control in manufacturing processes. Businesses that accurately classify images can streamline operations, reduce human error, and improve efficiency, resulting in cost savings and increased productivity.

Second, AI-powered image classification can improve user experiences on e-commerce platforms by providing personalized product recommendations based on visual preferences and browsing history. This not only increases customer satisfaction but also increases sales conversion rates for businesses, resulting in a win-win situation for both consumers and retailers.

Furthermore, in industrial settings, AI-powered image classification models can aid in predictive maintenance by detecting anomalies or defects in machinery components via visual inspection. This proactive approach can help to avoid costly downtimes and potential safety hazards, thereby increasing operational reliability and reducing risks.

Overall, this exercise shows how AI technologies, even in their most basic forms, have enormous potential to transform various aspects of business, consumer interactions, and industrial processes. Organizations that use AI for image classification tasks can unlock new opportunities for innovation, efficiency, and competitiveness in today's dynamic and increasingly digitalized landscape.

Appendix:

```
import tensorflow as tf
from keras import layers
from keras.applications import VGG16
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import cv2

# Load and preprocess data
(train_X,             train_y),             (test_X,             test_y)             =
tf.keras.datasets.cifar100.load_data(label_mode='fine')

# Map fine class names to their respective class indices
fine_class_names = ['apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed', 'bee',
'beetle', 'bicycle', 'bottle',
                    'bowl', 'boy', 'bridge', 'bus', 'butterfly', 'camel', 'can',
'castle', 'caterpillar', 'cattle',
                   'chair', 'chimpanzee', 'clock', 'cloud', 'cockroach', 'couch',
'crab', 'crocodile', 'cup', 'dinosaur',
                    'dolphin', 'elephant', 'flatfish', 'forest', 'fox', 'girl',
'hamster', 'house', 'kangaroo', 'keyboard',
                   'lamp', 'lawn_mower', 'leopard', 'lion', 'lizard', 'lobster',
'man', 'maple_tree', 'motorcycle', 'mountain',
                   'mouse', 'mushroom', 'oak_tree', 'orange', 'orchid', 'otter',
'palm_tree', 'pear', 'pickup_truck', 'pine_tree',
                    'plain', 'plate', 'poppy', 'porcupine', 'possum', 'rabbit',
'raccoon', 'ray', 'road', 'rocket', 'rose',
                  'sea', 'seal', 'shark', 'shrew', 'skunk', 'skyscraper', 'snail',
'snake', 'spider', 'squirrel', 'streetcar',
```

```
                        'sunflower', 'sweet_pepper', 'table', 'tank', 'telephone',
'television', 'tiger', 'tractor', 'train', 'trout',
                        'tulip', 'turtle', 'wardrobe', 'whale', 'willow_tree', 'wolf',
'woman', 'worm']

# Filter images for the coarse class "fish" and the selected fine classes
selected_fine_classes = ['aquarium_fish', 'flatfish', 'ray', 'shark', 'trout']
selected_fine_indices = [fine_class_names.index(cls) for cls in
selected_fine_classes]
train_mask = np.isin(train_y.flatten(), selected_fine_indices)
test_mask = np.isin(test_y.flatten(), selected_fine_indices)
train_X, train_y = train_X[train_mask], train_y[train_mask]
test_X, test_y = test_X[test_mask], test_y[test_mask]

# Update label mappings for the selected fine classes
label_mapping = {selected_fine_indices[i]: i for i in
range(len(selected_fine_indices))}
train_y = np.vectorize(label_mapping.get)(train_y)
test_y = np.vectorize(label_mapping.get)(test_y)

# Normalize pixel values to [0, 1]
train_X, test_X = train_X / 255.0, test_X / 255.0

# Resize images to match VGG16 input size
train_X_resized = np.array([cv2.resize(img, (224, 224)) for img in train_X])
test_X_resized = np.array([cv2.resize(img, (224, 224)) for img in test_X])

# Load pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
for layer in base_model.layers:
    layer.trainable = False

# Build model
model = tf.keras.Sequential([
    base_model,
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.5),
```

```
    layers.Dense(len(selected_fine_classes), activation='softmax')
])

# Compile model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Learning Rate Scheduling
lr_scheduler = tf.keras.callbacks.ReduceLROnPlateau(factor=0.5, patience=3)

# Train model
history = model.fit(train_X_resized, train_y, epochs=20,
validation_data=(test_X_resized, test_y), callbacks=[lr_scheduler])

# Evaluate model
test_loss, test_acc = model.evaluate(test_X_resized, test_y)
print(f'Test accuracy: {test_acc * 100:.2f}%')

# Confusion matrix
predictions = np.argmax(model.predict(test_X_resized), axis=1)
cm = confusion_matrix(test_y, predictions)
print(cm)

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()

# Visualize predictions
class_names = selected_fine_classes
plt.figure(figsize=(12, 12))
for i in range(36):
    plt.subplot(6, 6, i + 1)
```

```
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(test_X_resized[i], cmap=plt.cm.binary)
    true_label = class_names[test_y[i][0]]
    pred_label = class_names[predictions[i]]
    plt.xlabel(f"True: {true_label}\nPred: {pred_label}")
plt.tight_layout()
plt.show()
```