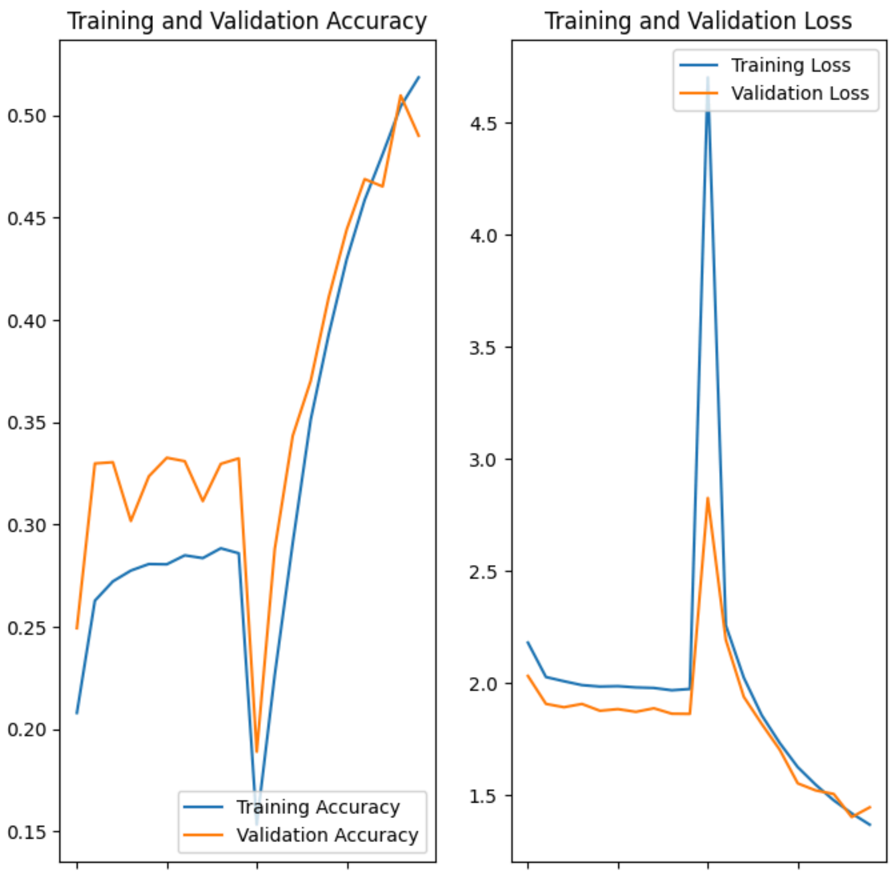
MADHUR GUPTA

AIM - To fine tune a pre-trained CNN architecture and evaluate its performance on a dataset.

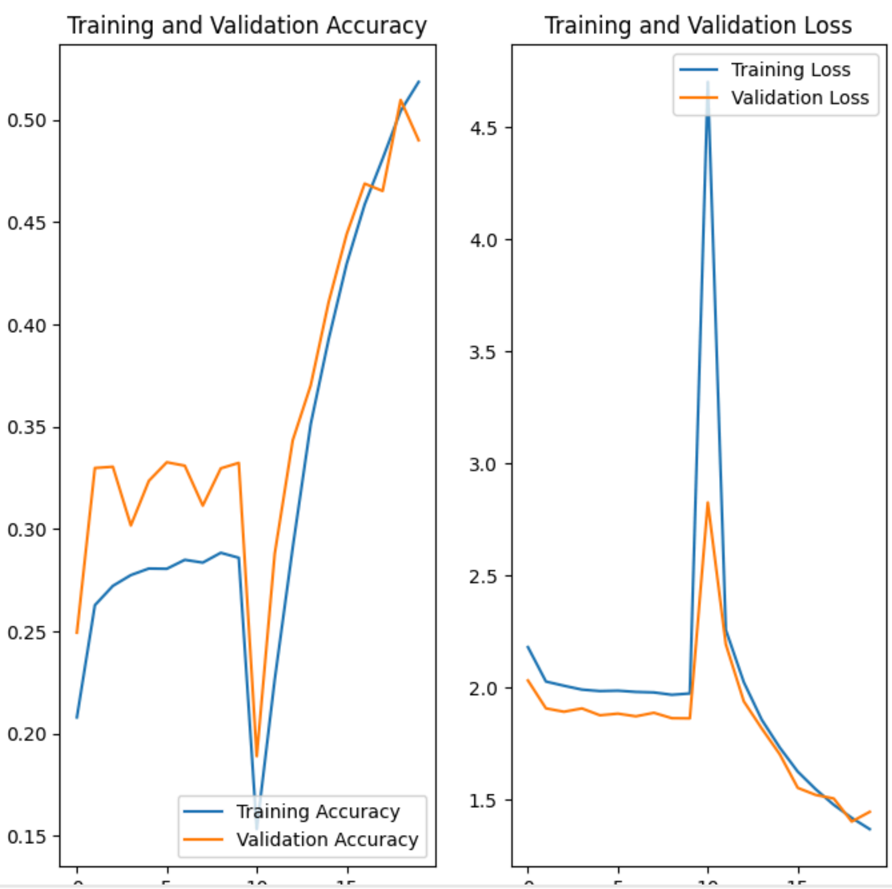
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import numpy as np

# Load and preprocess the dataset
# Replace with your actual dataset loading and preprocessing
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
y_train = keras.utils.to_categorical(y_train, num_classes=10)
y_test = keras.utils.to_categorical(y_test, num_classes=10)

# Define the pre-trained CNN model
base_model = keras.applications.ResNet50(
    weights="imagenet", include_top=False, input_shape=(32, 32, 3)
)

# Freeze the pre-trained layers
base_model.trainable = False

# Add custom classification layers
inputs = keras.Input(shape=(32, 32, 3))
x = base_model(inputs, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.2)(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs, outputs)

# Compile the model
model.compile(
    optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"]
)

# Train the model
epochs = 10 # Adjust as needed
history = model.fit(x_train, y_train, epochs=epochs, validation_data=(x_test, y_test))


# Fine-tune the top layers of the pre-trained model
base_model.trainable = True
for layer in base_model.layers[:100]: # Fine-tune a specific number of layers
    layer.trainable = False

model.compile(
    optimizer=keras.optimizers.Adam(1e-5),  # Low learning rate for fine-tuning
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)

fine_tune_epochs = 10 # Adjust as needed
total_epochs =  epochs + fine_tune_epochs
history_fine = model.fit(x_train, y_train, epochs=total_epochs, initial_epoch=epochs, validation_data=(x_test, y_test))

# Evaluate the model
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")


#Plot training history
acc = history.history['accuracy'] + history_fine.history['accuracy']
val_acc = history.history['val_accuracy'] + history_fine.history['val_accuracy']
loss = history.history['loss'] + history_fine.history['loss']
val_loss = history.history['val_loss'] + history_fine.history['val_loss']

epochs_range = range(total_epochs)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_orderi
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 42s 18ms/step - accuracy: 0.1695 - loss: 2.3354 - val_accuracy: 0.2494 - val_loss: 2.0318
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 22s 9ms/step - accuracy: 0.2568 - loss: 2.0350 - val_accuracy: 0.3299 - val_loss: 1.9075
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 14s 9ms/step - accuracy: 0.2724 - loss: 2.0110 - val_accuracy: 0.3305 - val_loss: 1.8925
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 15s 9ms/step - accuracy: 0.2766 - loss: 1.9968 - val_accuracy: 0.3018 - val_loss: 1.9073
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 11ms/step - accuracy: 0.2799 - loss: 1.9830 - val_accuracy: 0.3236 - val_loss: 1.8769
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 18s 9ms/step - accuracy: 0.2836 - loss: 1.9831 - val_accuracy: 0.3327 - val_loss: 1.8841
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 9ms/step - accuracy: 0.2848 - loss: 1.9760 - val_accuracy: 0.3310 - val_loss: 1.8721
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 15s 9ms/step - accuracy: 0.2834 - loss: 1.9782 - val_accuracy: 0.3115 - val_loss: 1.8879
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 15s 9ms/step - accuracy: 0.2875 - loss: 1.9748 - val_accuracy: 0.3297 - val_loss: 1.8637
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 15s 9ms/step - accuracy: 0.2846 - loss: 1.9802 - val_accuracy: 0.3324 - val_loss: 1.8630
Epoch 11/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 87s 35ms/step - accuracy: 0.1380 - loss: 7.5832 - val_accuracy: 0.1890 - val_loss: 2.8255
Epoch 12/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 34s 22ms/step - accuracy: 0.2098 - loss: 2.3339 - val_accuracy: 0.2880 - val_loss: 2.1933
Epoch 13/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 40s 21ms/step - accuracy: 0.2756 - loss: 2.0669 - val_accuracy: 0.3434 - val_loss: 1.9389
Epoch 14/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 34s 22ms/step - accuracy: 0.3400 - loss: 1.8901 - val_accuracy: 0.3703 - val_loss: 1.8178
Epoch 15/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 40s 21ms/step - accuracy: 0.3838 - loss: 1.7544 - val_accuracy: 0.4111 - val_loss: 1.7029
Epoch 16/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 33s 21ms/step - accuracy: 0.4248 - loss: 1.6395 - val_accuracy: 0.4442 - val_loss: 1.5532
Epoch 17/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 41s 21ms/step - accuracy: 0.4579 - loss: 1.5554 - val_accuracy: 0.4689 - val_loss: 1.5216
Epoch 18/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 33s 21ms/step - accuracy: 0.4805 - loss: 1.4817 - val_accuracy: 0.4653 - val_loss: 1.5058
Epoch 19/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 41s 21ms/step - accuracy: 0.5037 - loss: 1.4207 - val_accuracy: 0.5098 - val_loss: 1.4029
Epoch 20/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 42s 21ms/step - accuracy: 0.5151 - loss: 1.3738 - val_accuracy: 0.4901 - val_loss: 1.4461
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.4893 - loss: 1.4455
Test Loss: 1.446104884147644
Test Accuracy: 0.4900999963283539
```




```
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
```

```
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



Start coding or generate with AI.