

Kevin Tong - Bird Image Classification

Packages

```
In [73]: import datetime
import time
from packaging import version
from collections import Counter
import numpy as np
import pandas as pd

import matplotlib as mpl # EA
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import confusion_matrix, classification_report
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.models import Sequential
import tensorflow.keras.backend as z
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import Conv2D, MaxPool2D, BatchNormalization, Dropout, Flatten, Dense
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.preprocessing import image
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dropout
import os
import cv2
from PIL import Image
from tensorflow.keras.regularizers import l1
```

```
In [2]: # from google.colab import drive
# drive.mount('/content/drive')
data_dir = '/content/drive/MyDrive/Northwestern MSDS/MSDS 458/458_Final_Notebooks/spectrogram_output'
```

Preprocessing

```
In [3]: # Initialize a set to store unique classes (bird species)
unique_classes = set()

# Define a function to load and preprocess images
def load_and_preprocess_images(image_paths):
    images = []
    for path in image_paths:
        img = Image.open(path)
        img = img.resize((224, 224)) # Resize to your desired dimensions
        img = img.convert("RGB") # Convert to RGB (remove alpha channel if present)
        img = np.array(img) / 255.0 # Normalize pixel values
        images.append(img)
    return np.array(images)

# Loop through directories and load images
all_images = []
for bird_folder in os.listdir(data_dir):
    bird_folder_path = os.path.join(data_dir, bird_folder)
    if os.path.isdir(bird_folder_path) and len(os.listdir(bird_folder_path)) > 1:
        # Add the current folder name to the set of unique classes
        unique_classes.add(bird_folder)

        image_paths = [os.path.join(bird_folder_path, img) for img in os.listdir(bird_folder_path)]
        images = load_and_preprocess_images(image_paths)
        all_images.extend(images)

all_images = np.array(all_images)

# Calculate the total number of classes
num_classes = len(unique_classes)
print("Number of classes:", num_classes)
```

Number of classes: 110

```
In [4]: # Load and preprocess labels
all_labels = []
for bird_folder in os.listdir(data_dir):
    bird_folder_path = os.path.join(data_dir, bird_folder)
    if os.path.isdir(bird_folder_path) and len(os.listdir(bird_folder_path)) > 1:
        label = bird_folder # Assuming folder name is the label
        labels = [label] * len(os.listdir(bird_folder_path))
        all_labels.extend(labels)

# Convert labels to numerical format (e.g., one-hot encoding)
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(all_labels)
onehot_encoder = OneHotEncoder(sparse=False)
encoded_labels = onehot_encoder.fit_transform(integer_encoded.reshape(-1, 1))

/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
warnings.warn()
```

```
In [5]: from sklearn.model_selection import train_test_split

# Split data into training and temporary set (which includes validation and test data)
train_images, temp_images, train_labels, temp_labels = train_test_split(all_images, encoded_labels, test_size=0.2, random_state=42)

# Split temporary set into validation and test sets
val_images, test_images, val_labels, test_labels = train_test_split(temp_images, temp_labels, test_size=0.5, random_state=42)
```

```
In [6]: print("Train images shape:", train_images.shape)
print("Validation images shape:", val_images.shape)
print("Test images shape:", test_images.shape)
```

Train images shape: (1725, 224, 224, 3)
Validation images shape: (216, 224, 224, 3)
Test images shape: (216, 224, 224, 3)

Neural Network

```
In [74]: start_time = time.time()

# Define your CNN model with Conv2D and Dropout layers
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(256, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),

    # Add dropout layers
    layers.Dropout(0.5), # Add dropout after flattening
    layers.Dense(128, activation='relu', kernel_regularizer=l1(0.01)), # L1 regularization

    layers.Dropout(0.5), # Add dropout after the first dense layer
    layers.Dense(num_classes, activation='softmax') # num_classes is the number of bird species
])

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

```
In [75]: # Print the model summary
model.summary()
```

Model: "sequential_12"

Layer (type)	Output Shape	Param #
conv2d_21 (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d_21 (MaxPoolin g2D)	(None, 111, 111, 32)	0
conv2d_22 (Conv2D)	(None, 109, 109, 64)	18496
max_pooling2d_22 (MaxPoolin g2D)	(None, 54, 54, 64)	0
conv2d_23 (Conv2D)	(None, 52, 52, 128)	73856
max_pooling2d_23 (MaxPoolin g2D)	(None, 26, 26, 128)	0
conv2d_24 (Conv2D)	(None, 24, 24, 256)	295168
max_pooling2d_24 (MaxPoolin g2D)	(None, 12, 12, 256)	0
flatten_12 (Flatten)	(None, 36864)	0
dropout_18 (Dropout)	(None, 36864)	0
dense_30 (Dense)	(None, 128)	4718720
dropout_19 (Dropout)	(None, 128)	0
dense_31 (Dense)	(None, 110)	14190

=====
Total params: 5,121,326
Trainable params: 5,121,326
Non-trainable params: 0

```
In [76]: # Train the model
history = model.fit(train_images, train_labels, epochs=200, batch_size=64,
                    validation_data=(val_images, val_labels), callbacks=[
                        tf.keras.callbacks.ModelCheckpoint("CNN_model.h5", save_best_only=True, save_weights_only=False)
                        ,tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3),
                    ])

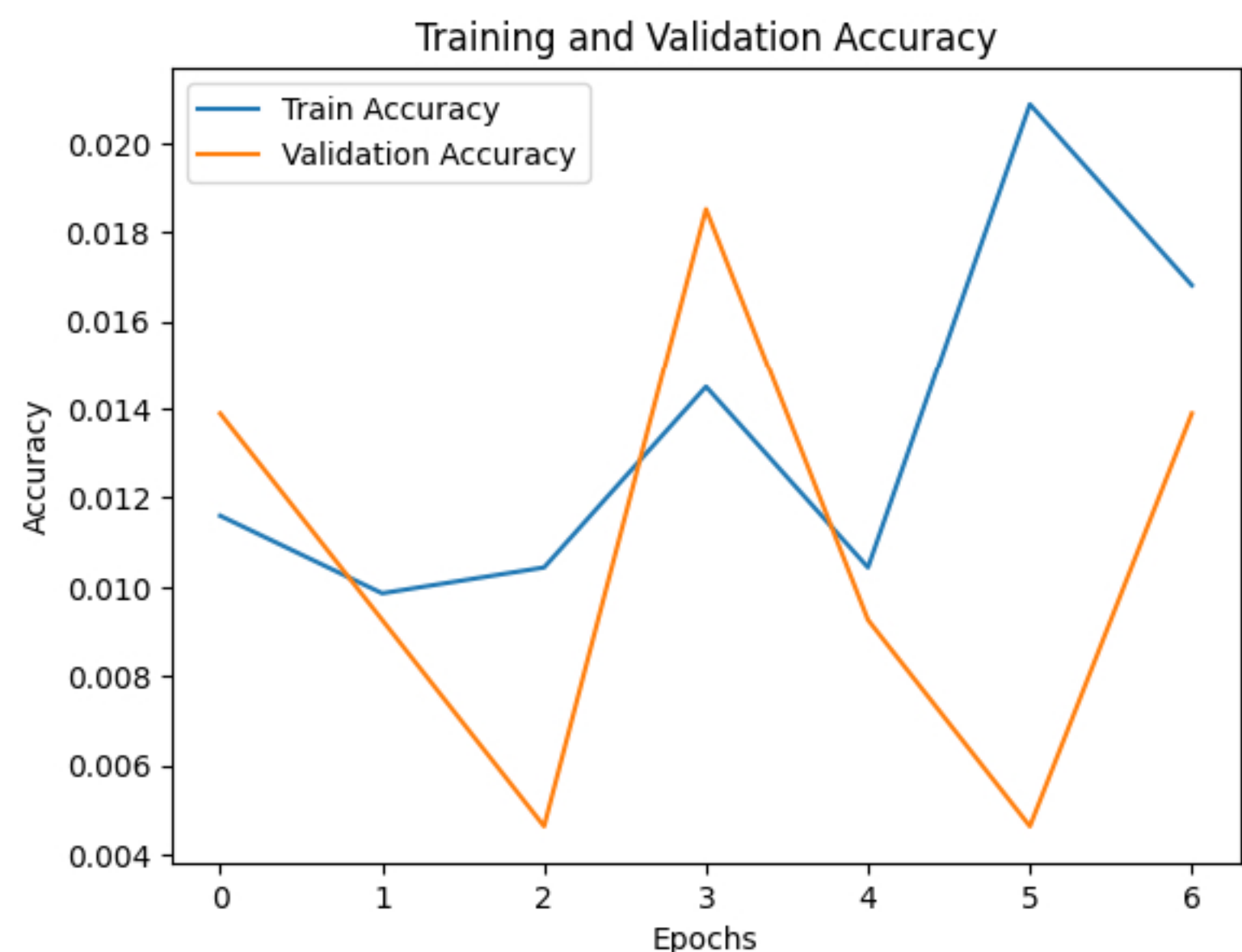
# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

Epoch 1/200
27/27 [=====] - 4s 48ms/step - loss: 84.2146 - accuracy: 0.0116 - val_loss: 25.7568 - val_a
ccuracy: 0.0139
Epoch 2/200
27/27 [=====] - 1s 33ms/step - loss: 16.3848 - accuracy: 0.0099 - val_loss: 12.4966 - val_a
ccuracy: 0.0093
Epoch 3/200
27/27 [=====] - 1s 33ms/step - loss: 11.5558 - accuracy: 0.0104 - val_loss: 11.2174 - val_a
ccuracy: 0.0046
Epoch 4/200
27/27 [=====] - 1s 32ms/step - loss: 11.1806 - accuracy: 0.0145 - val_loss: 11.0224 - val_a
ccuracy: 0.0185
Epoch 5/200
27/27 [=====] - 1s 33ms/step - loss: 10.9890 - accuracy: 0.0104 - val_loss: 10.8509 - val_a
ccuracy: 0.0093
Epoch 6/200
27/27 [=====] - 1s 27ms/step - loss: 10.7355 - accuracy: 0.0209 - val_loss: 10.8907 - val_a
ccuracy: 0.0046
Epoch 7/200
27/27 [=====] - 1s 33ms/step - loss: 10.6481 - accuracy: 0.0168 - val_loss: 10.5785 - val_a
ccuracy: 0.0139
7/7 [=====] - 0s 9ms/step - loss: 10.5518 - accuracy: 0.0093
Test accuracy: 0.009259259328246117

```
In [77]: end_time = time.time()
runtime = end_time - start_time
print('Total runtime:', runtime, 'seconds')
```

Total runtime: 13.12798023223877 seconds

```
In [78]: # Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()
```



```
In [79]: # Plot test accuracy
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(len(history.history['val_accuracy']), test_acc, 'ro', label='Test Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Test Accuracy')
plt.legend()
plt.show()
```

