airplaneVSbird

March 2, 2025

1 Classifying Airplanes vs. Birds Using Deep Learning

1.1 Introduction

This project is an example of computer vision, which is a field of AI where machine learning models interpret and understand *visual* data. This project implements a binary image classification model to distinguish between airplanes and birds. It uses the CIFAR-10 database, which consists of 60000 imaged from 10 different classes; airplanes and birds being 2 of those categories. I chose this project as it's easy access to pretrained data that may be useful/similar to skills needed over the course of this internship (Airplanes and birds kinda look like drones, right?)

To achieve high classification accuracy in a pinch, I used transfer learning using MobileNetV2, which is a pre-trained model for image classification. The final model is fine tuned on the bird and plane dataset, and evaluated using various performance metrics; accuracy curves, loss curves, confusion matrix, and sample predictions.

1.2 Imports

[1]:
```
import tensorflow as tf
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
```

```
2025-02-25 03:45:05.043248: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
/Users/alyviaconey/venv/lib/python3.9/site-packages/urllib3/__init__.py:35:
NotOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl'
module is compiled with 'LibreSSL 2.8.3'. See:
https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
```

1.3 Loading and Preprocessing Data

Here the airplane and birds classes from the cifar-10 dataset are loaded in. They are resized to 128x128 pixels because this is required to use MobileNetV2. Pixel values are also norrmalized to a

range of [0, 1] for faster convergence. Labels are also changed to binary, where 1 is bird and 0 is airplane. Train-test split: 80:20

[3]:
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
classes = [0, 2]

# Filter dataset for only airplanes (0) and birds (2)
train_filter = (y_train.flatten() == classes[0]) | (y_train.flatten() ==␣
 ↪classes[1])
test_filter = (y_test.flatten() == classes[0]) | (y_test.flatten() ==␣
 ↪classes[1])

x_train, y_train = x_train[train_filter], y_train[train_filter]
x_test, y_test = x_test[test_filter], y_test[test_filter]

# Normalize and resize images
x_train = tf.image.resize(x_train, (128, 128)).numpy() / 255.0
x_test = tf.image.resize(x_test, (128, 128)).numpy() / 255.0

# Convert labels to binary (0 for airplane, 1 for bird)
y_train = (y_train == classes[1]).astype(int)
y_test = (y_test == classes[1]).astype(int)

print("Data Loaded & Preprocessed!")
```

```
Data Loaded & Preprocessed!
```

1.4 Defining the Model

Transfer learning: the pretrained model (MobileNetV2) is used as the feature extractor, shown in the code as include_top=False. The pretrained layers are "frozen" to prevent any modifications or changes to the learned ImageNet weights.

GlobalAveragePooling2D: processes output features by reducing spatial dimensions and extracting feature representations

Dropout (0.2): regularization to prevent overfitting the model

Dense Layer (Sigmoid Activation): a single output neuron (0 for airplane, 1 for bird) for binary classification

Sequential was used because in Keras, the model is a linear stack of layers, and the flow of information is straightforward from input to output. This is a binary classification task, but a sequential model with a single neuron and sigmoid activation effectively handles it.

[5]:
```
base_model = tf.keras.applications.MobileNetV2(
    input_shape=(128, 128, 3), include_top=False, weights="imagenet"
)
base_model.trainable = False  # Freeze pretrained weights
```

```
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam", loss="binary_crossentropy",␣
 ↪metrics=["accuracy"])

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
mobilenetv2_1.00_128 (Functional)	(None, 4, 4, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 1)	1,281

Total params: 2,259,265 (8.62 MB)

Trainable params: 1,281 (5.00 KB)

Non-trainable params: 2,257,984 (8.61 MB)

1.5 Train the model

Data augmentation improves the generalization of the model. Ex) here the data is flipped and rotated. This way the model has a higher chance of recognizing flipped images or images without standard orientation.

Binary Crossentropy Loss is just the loss function that is used for binary classification problems. It is good to use because it more strongly penalizes wrong predictions, and ensures the model outputs a probability for the sigmoid activation function (which is required for Sequential models).

Adam Optimizer is used for adaptive learning rates

The model is trained for 10 epochs with a batch size of 32. More epochs could improve the accuracy, but in the interest of time and demonstration, 10 were used.

[30]:
```
# Data Augmentation
datagen = tf.keras.preprocessing.image.ImageDataGenerator(rotation_range=20,
          horizontal_flip=True)

# Train Model
history = model.fit(
    datagen.flow(x_train, y_train, batch_size=32),
    validation_data=(x_test, y_test),
    epochs=10
)
```

```
/Users/alyviaconey/venv/lib/python3.9/site-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()

Epoch 1/10
 92/313             50s 229ms/step -
accuracy: 0.9349 - loss: 0.1591
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[30], line 6
      2 datagen = tf.keras.preprocessing.image.
 ↪ImageDataGenerator(rotation_range=20,
      3         horizontal_flip=True)
      5 # Train Model
----> 6 history = model.fit(
      7     datagen.flow(x_train, y_train, batch_size=32),
      8     validation_data=(x_test, y_test),
      9     epochs=10
     10 )

File ~/venv/lib/python3.9/site-packages/keras/src/utils/traceback_utils.py:117,
 ↪in filter_traceback.<locals>.error_handler(*args, **kwargs)
    115 filtered_tb = None
    116 try:
--> 117     return fn(*args, **kwargs)
    118 except Exception as e:
    119     filtered_tb = _process_traceback_frames(e.__traceback__)
```

```
    261     )

File ~/venv/lib/python3.9/site-packages/tensorflow/python/eager/context.py:1500
 ↪in Context.call_function(self, name, tensor_inputs, num_outputs)
   1498 cancellation_context = cancellation.context()
   1499 if cancellation_context is None:
-> 1500   outputs = execute.execute(
   1501       name.decode("utf-8"),
   1502       num_outputs=num_outputs,
   1503       inputs=tensor_inputs,
   1504       attrs=attrs,
   1505       ctx=self,
   1506   )
   1507 else:
   1508   outputs = execute.execute_with_cancellation(
   1509       name.decode("utf-8"),
   1510       num_outputs=num_outputs,
   (…)
   1514       cancellation_manager=cancellation_context,
   1515   )

File ~/venv/lib/python3.9/site-packages/tensorflow/python/eager/execute.py:53,
 ↪in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     51 try:
     52   ctx.ensure_initialized()
---> 53   tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name
     54                                       inputs, attrs, num_outputs)
     55 except core._NotOkStatusException as e:
     56   if name is not None:

KeyboardInterrupt:
```

1.6 Save Model

Can ignore, this is just so that the model doesn't need to be retrained every time this notebook is used.

[9]:
```
model.save("/Users/alyviaconey/Desktop/plane_vs_bird_model.keras")
#use the line below to load model later
#model = tf.keras.models.load_model("/Users/alyviaconey/Desktop/
 ↪plane_vs_bird_model.keras")

print("Model Saved Successfully!")
```

```
Model Saved Successfully!
```

[23]:
```
#### Ignore - for visualizations
```

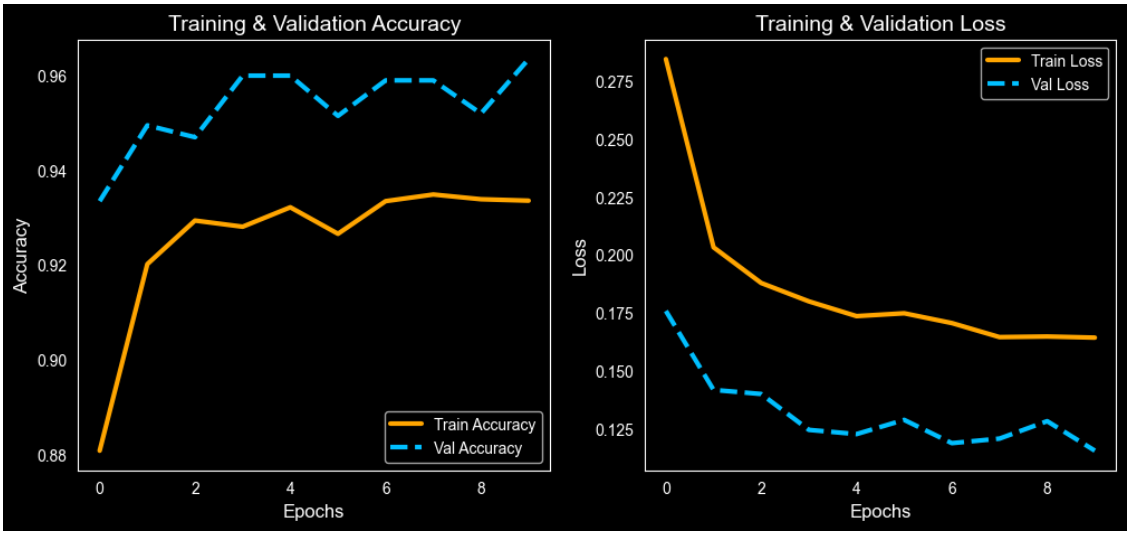
1.7 Accuracy and Loss

[28]:
```
plt.figure(figsize=(12, 5))

# Accuracy Plot
plt.subplot(1, 2, 1)
plt.plot(history.history["accuracy"], label="Train Accuracy",␣
 ↪color=neon_orange, linewidth=3)
plt.plot(history.history["val_accuracy"], label="Val Accuracy",␣
 ↪color=light_blue, linestyle="dashed", linewidth=3)
plt.xlabel("Epochs", fontsize=12, color="white")
plt.ylabel("Accuracy", fontsize=12, color="white")
plt.legend(fontsize=10)
plt.title("Training & Validation Accuracy", fontsize=14, color="white")

# Loss Plot
plt.subplot(1, 2, 2)
plt.plot(history.history["loss"], label="Train Loss", color=neon_orange,␣
 ↪linewidth=3)
plt.plot(history.history["val_loss"], label="Val Loss", color=light_blue,␣
 ↪linestyle="dashed", linewidth=3)
plt.xlabel("Epochs", fontsize=12, color="white")
plt.ylabel("Loss", fontsize=12, color="white")
plt.legend(fontsize=10)
plt.title("Training & Validation Loss", fontsize=14, color="white")

# Show plots
plt.show()
```



Interpreting the results If the training accuracy is high, but the validation (test) accuracy is low, the model is overfitting. Here we can see that the test accuracy is actually slightly higher than the training accuracy, so our model is not overfitting the data.

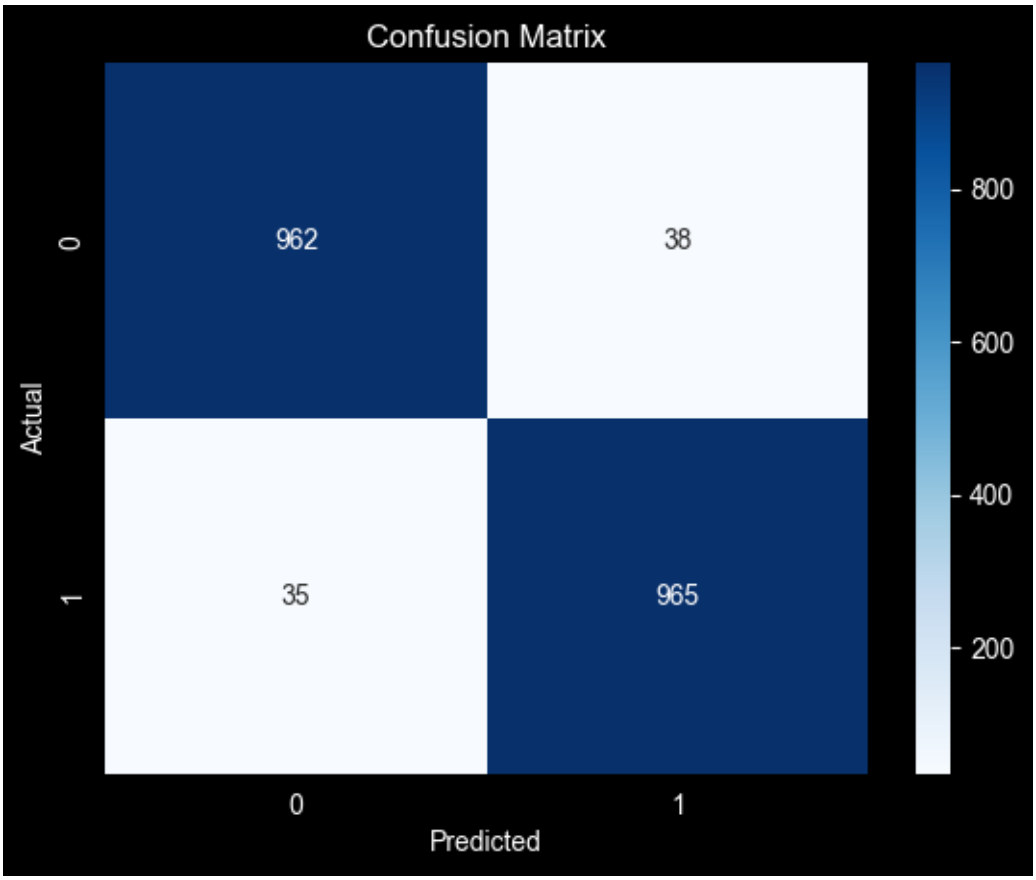
For the second chart, a gradual decrease in loss indicates good convergence, which a gradual decrease is seen in the loss of both the training and test data.

1.8 Confusion Matrix

[29]:
```
y_pred = (model.predict(x_test) > 0.5).astype(int)

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
63/63              13s 198ms/step
```



Interpreting the results A confusion matrix basically is a tool for showing how well a model performs. It just shows the amount of times the model predicted correctly and incorrectly. Here we can see the dark blue boxes represent correct predictions, and the lighter, almost white, colored boxes are the number of miscalculations. Based on the confusion matric, this model is pretty strong.

1.9 Sample Prediction

This is just a visual confirmation that the model makes correct predictions

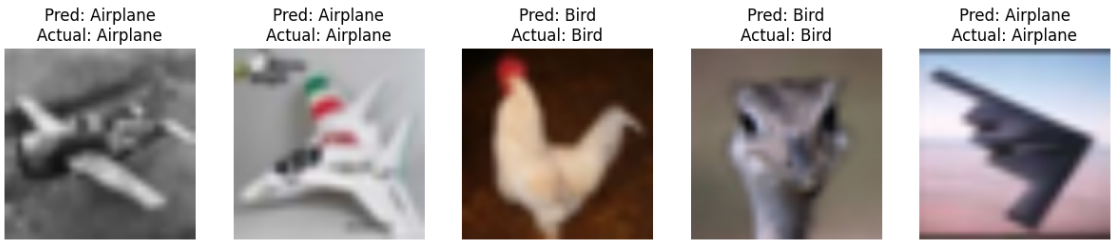
[14]:
```
import random

num_images = 5
indices = random.sample(range(len(x_test)), num_images)
images = x_test[indices]
labels = y_test[indices]

preds = (model.predict(images) > 0.5).astype(int)

plt.figure(figsize=(15, 5))
for i, img in enumerate(images):
    plt.subplot(1, num_images, i+1)
    plt.imshow(img)
    plt.title(f"Pred: {'Bird' if preds[i] else 'Airplane'}\nActual: {'Bird' if␣
 ↪labels[i] else 'Airplane'}")
    plt.axis("off")
plt.show()
```

```
1/1             0s 90ms/step
```



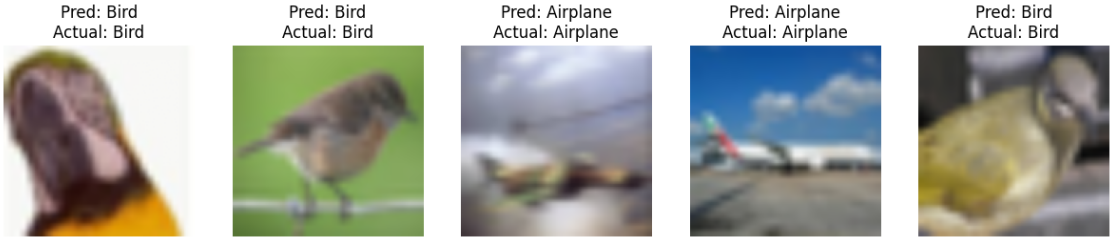
[22]:
```
indices = random.sample(range(len(x_test)), num_images)
images = x_test[indices]
labels = y_test[indices]

preds = (model.predict(images) > 0.5).astype(int)

plt.figure(figsize=(15, 5))
for i, img in enumerate(images):
    plt.subplot(1, num_images, i+1)
```

```
    plt.imshow(img)
    plt.title(f"Pred: {'Bird' if preds[i] else 'Airplane'}\nActual: {'Bird' if␣
 ↪labels[i] else 'Airplane'}")
    plt.axis("off")
plt.show()
```

```
1/1             0s 78ms/step
```



1.10 Future Improvements

- Increase the epochs for better convergence (when the loss stabilizes and the accuracy plateaus).
- The dataset size could be larger for better generalization

1.11 Conclusion

The model achieves *>93% accuracy* with MobileNetV2, which could be good depending on the context this model is used in. If we're using this model for a technology to decide to take some action upon planes and leave birds alone, errors, especially false positives, could be incredibly harmful.

Data augmentation and transfer learning boosted the performance of the model. There are other improvements that could be made and this project is fairly introductory, but it demonstrates effective use of computer vision. The high accuracy achieved by this model shows how powerful deep learning based computer vision systems can be even with minimal training.