Nike Shoe Classifier - Project Documentation

Project Overview:

The Nike Shoe Classifier is an AI-based image classification model deployed as a FastAPI web service on Google Cloud Run. It allows users to upload images and classify them as:

- Nike
- Other Shoes
- Non-Shoe Items

The model is trained on a dataset containing Nike and non-Nike sneakers, along with non-shoe images to improve robustness.

Key Features:

❖ Multi-class classification (Nike, Other, Non-Shoe)
❖ Deployed on Google Cloud Run for scalability
❖ FastAPI with automatic Swagger UI (/docs)
❖ Dockerized for easy deployment
❖ Accessible via API for integration with web/mobile apps

Technologies Used:

Technology	Purpose
Python 3.9	Programming language
TensorFlow/Keras	Model training & inference
FastAPI	Web API framework
Uvicorn	ASGI server to run FastAPI
Docker	Containerization for deployment
Google Cloud Run	Serverless hosting
Swagger UI	Auto-generated API documentation
Google Container Registry	Stores Docker images

Project Structure:

```
/NikeClassifier
            │── /app
            │   │── model.py       # Model loading & prediction functions
            │   │── __init__.py    # Package initialization
            │   │── main.py        # FastAPI application
            │   │── utils.py       # Helper functions (image preprocessing)
            │   │── nike_final_model.keras                 #Model used for predictions
            │── Dockerfile         # Docker setup for deployment
            │── /Notebooks
            │   │── Nikedifferentapproach.ipynb          #Notebook that contains the code for the model
            │── /Documentation
            │   │── Documentation.docx      #Documentation of the project
            │── requirements.txt    # Python dependencies
            │── run.sh             # Shell script for building & running Docker container
```

AI Model Documentation:

Project Overview

- **Problem Statement:** Develop an AI model that accurately classifies sneaker images into **Nike, Other Shoes, and Non-Shoe Items**.
- **Project Goals:** Improve sneaker classification accuracy for retail and authentication purposes.
- **Project Scope:** Limited to sneaker images; does not classify other apparel or brands outside the dataset.

Data Description

Data Sources

The dataset was compiled from multiple sources, including:

- **Kaggle:** Publicly available sneaker image datasets.
- **Web Scraping:** Scraped images from Nike and other shoe retailer websites.
- **Internal Datasets:** Manually collected images for better classification performance.
- **Open-source Repositories:** Leveraged sneaker classification datasets from research sources.

Data Preprocessing:

- Image resizing to **224x224** pixels
- Normalization (rescaling pixel values between 0-1)
- Data augmentation (flipping, zoom, rotation, brightness adjustment)

Data Split:

- Training set: **80%**
- Validation set: **20%**
- Testing set: **A New Dataset which is not trained by the model**

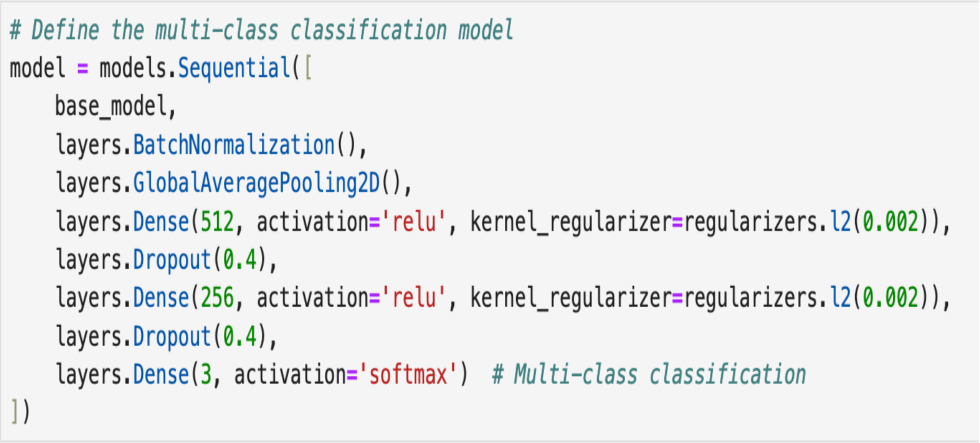
Data Quality Issues:

- Potential biases towards Nike branding features.
- Limited representation of rare sneaker models.

Model Details

Model Architecture:

- Base Model: MobileNetV2 (pre-trained on ImageNet)
- Batch Normalization: Normalizes activations to improve stability
- Global Average Pooling: Reduces dimensions while retaining information
- Fully Connected Layers:
  - 512 neurons, ReLU activation, L2 regularization (0.002)
  - 256 neurons, ReLU activation, L2 regularization (0.002)
- Dropout (0.4): Prevents overfitting
- Softmax Output Layer: Classifies into 3 categories (Nike, Other Shoes, Non-Shoe Items)
- Hyperparameters:
  - Optimizer: Adam (learning rate = 0.0003, decay applied) Loss Function: Categorical Crossentropy
  - Batch Size: 32
  - Epochs: 20 (initial), +10 fine-tuning

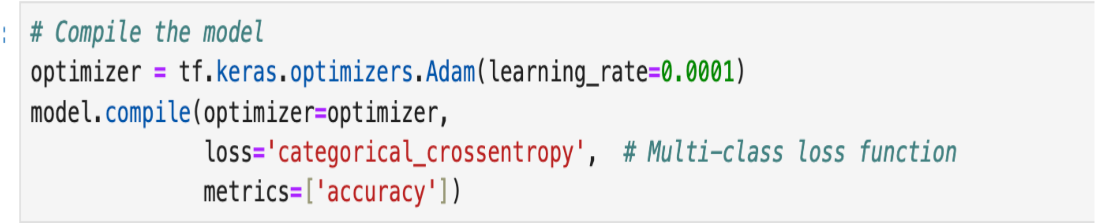


```
# Define the multi-class classification model
model = models.Sequential([
    base_model,
    layers.BatchNormalization(),
    layers.GlobalAveragePooling2D(),
    layers.Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.002)),
    layers.Dropout(0.4),
    layers.Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.002)),
    layers.Dropout(0.4),
    layers.Dense(3, activation='softmax')  # Multi-class classification
])
```

Training Process

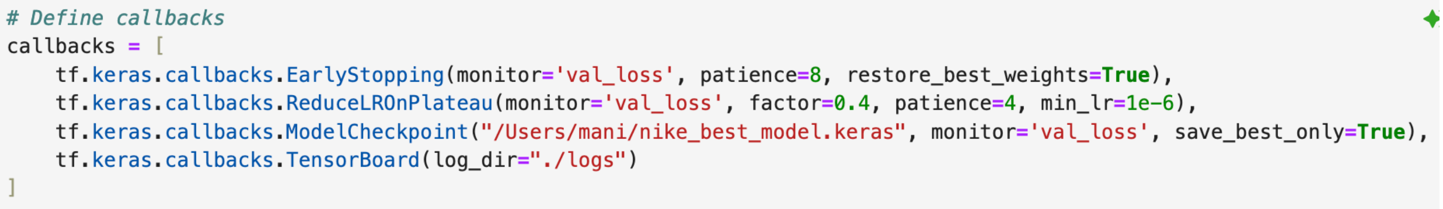
Training Overview

- The model is trained using transfer learning with MobileNetV2 as the base.
- The dataset is split into training, validation, and test sets.
- Data augmentation is applied to improve generalization and avoid overfitting.
- The training process is monitored using Early Stopping and ReduceLROnPlateau callbacks.
- Fine-tuning is performed after initial training to enhance feature learning.
- Validation Strategy:
  - Early stopping used to prevent overfitting.
  - ReduceLROnPlateau used to dynamically adjust learning rate.
- Training Metrics:
  - Accuracy, Loss, Learning Rate Adjustments.



```
# Compile the model
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',  # Multi-class loss function
              metrics=['accuracy'])
```

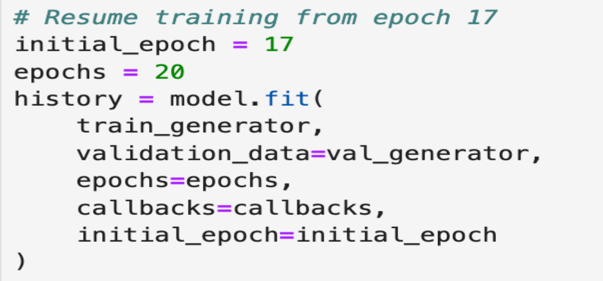
Callbacks Used



```
# Define callbacks
callbacks = [
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=8, restore_best_weights=True),
    tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.4, patience=4, min_lr=1e-6),
    tf.keras.callbacks.ModelCheckpoint("/Users/mani/nike_best_model.keras", monitor='val_loss', save_best_only=True),
    tf.keras.callbacks.TensorBoard(log_dir="./logs")
]
```

Initial Training

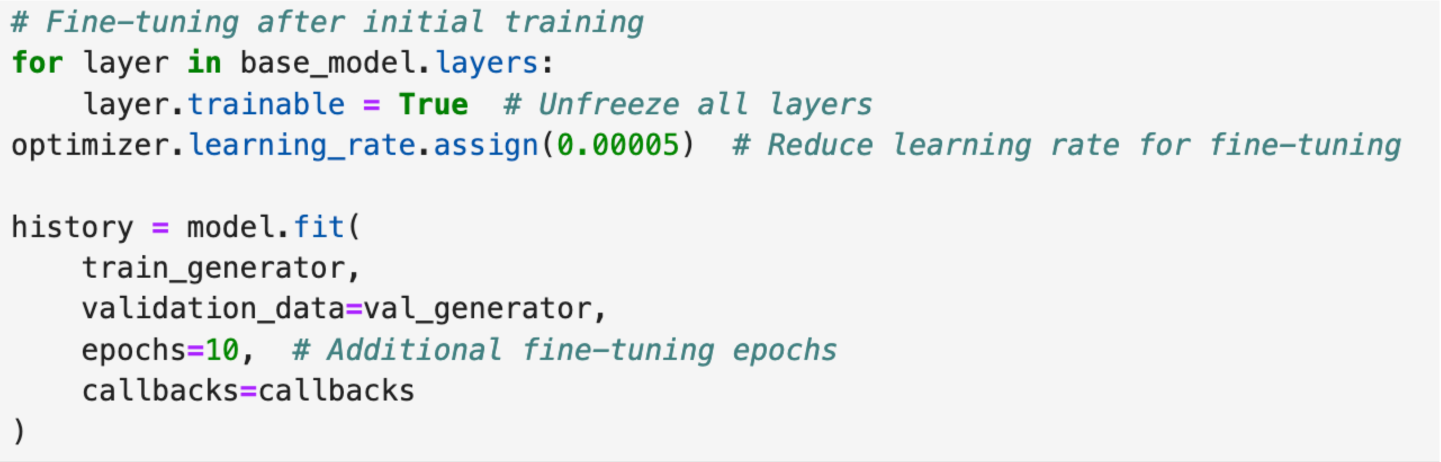
- The model is trained with frozen layers to extract generic features.
- It runs for **20 epochs**, with training resuming from epoch **17**.



```
# Resume training from epoch 17
initial_epoch = 17
epochs = 20
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=epochs,
    callbacks=callbacks,
    initial_epoch=initial_epoch
)
```

Fine-Tuning

- After initial training, all layers of the base model are unfrozen.
- The learning rate is reduced to **0.00005** for fine-tuning.
- Additional training is performed for **10 epochs** to refine feature learning.



```
# Fine-tuning after initial training
for layer in base_model.layers:
    layer.trainable = True  # Unfreeze all layers
optimizer.learning_rate.assign(0.00005)  # Reduce learning rate for fine-tuning

history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=10,  # Additional fine-tuning epochs
    callbacks=callbacks
)
```

Bias Mitigation Strategies

To ensure fairness and reduce dataset bias, the following steps were taken:

- **Balanced Dataset Composition:** Efforts were made to collect equal representations of Nike, Other Shoes, and Non-Shoe images.
- **Augmentation Techniques:** Applied rotation, flipping, and color shifting to prevent over-reliance on specific visual patterns.
- **Oversampling & Undersampling:** Adjusted dataset ratios to balance underrepresented categories.
- **Bias Testing:** Evaluated predictions on diverse image sources to identify and correct skewed classifications.
- **Ongoing Monitoring:** Model retraining plans include continuous evaluation against new datasets to ensure bias reduction.

Key Insights from Classification Report

Nike Class

- **Precision:** 89% (Out of all Nike predictions, 89% were correct)
- **Recall:** 90% (Out of all actual Nike samples, 90% were identified correctly)

Other Class

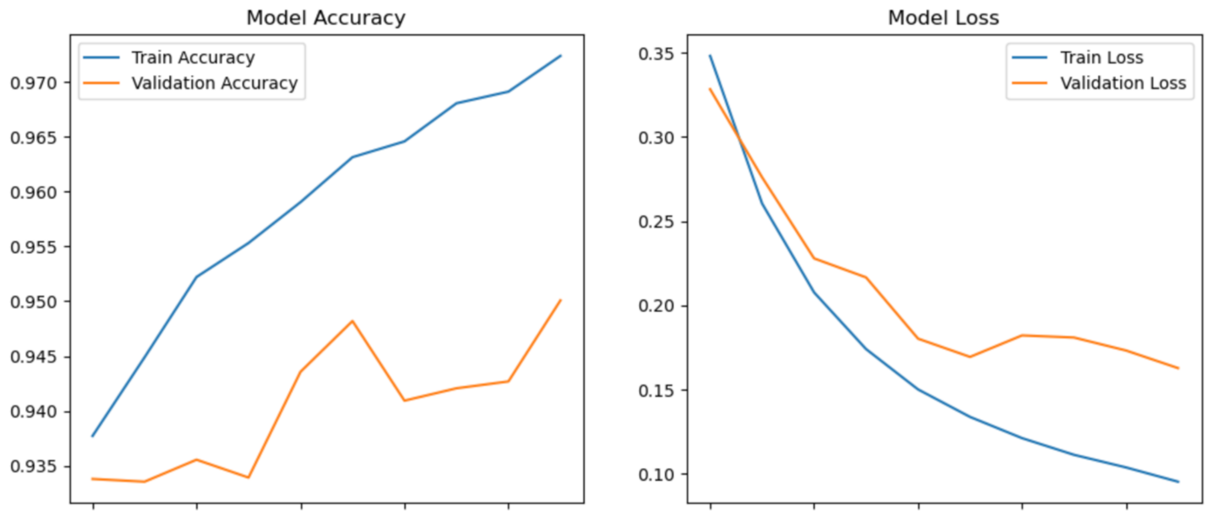
- **Precision: 91%**
- **Recall: 89%**

Non-Shoe Class

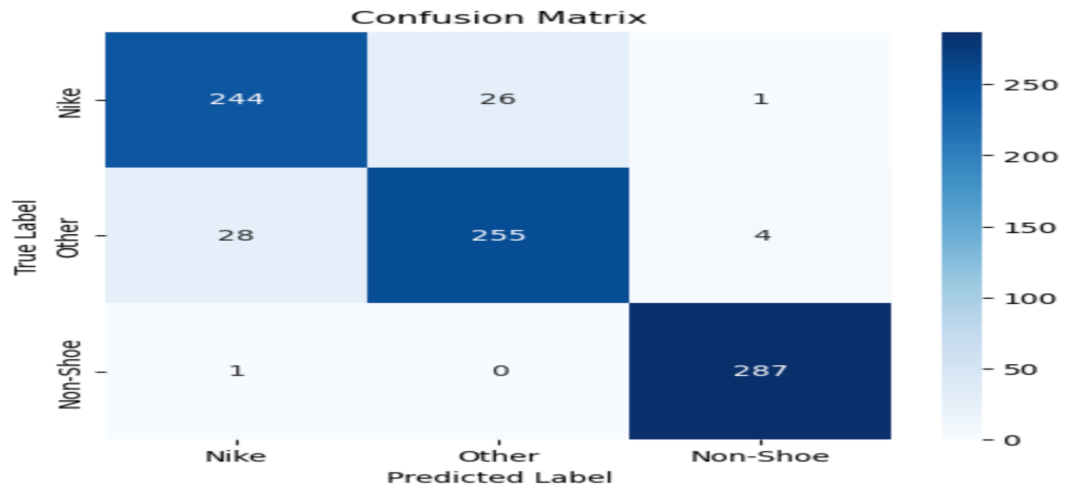
- Excellent classification with 98% Precision and 100% Recall, meaning nearly all Non-Shoe images were classified correctly.

Overall Model Performance

- **Accuracy:** 93% – A strong performance across all categories.
- **Balanced Precision, Recall, and F1-Scores** indicate reliable classification.
- **The model performs exceptionally well on distinguishing shoes from non-shoes** but has minor misclassification between Nike and Other brands.



```
Model training and evaluation complete.
```



```
              precision    recall  f1-score   support

        Nike       0.89      0.90      0.90       271
       Other       0.91      0.89      0.90       287
    Non-Shoe       0.98      1.00      0.99       288

    accuracy                           0.93       846
   macro avg       0.93      0.93      0.93       846
weighted avg       0.93      0.93      0.93       846
```

Potential Biases and Limitations

Bias Analysis:

- More Nike sneaker images in training data could lead to overfitting.
- Dark-colored shoes occasionally misclassified due to limited contrast variations.

Limitations:

- Performance may degrade with unseen sneaker models not in training set.
- Requires clear images; low-resolution or occluded images reduce accuracy.

How the Model Works:

- User uploads an image via the API (/predict).
- Image is preprocessed (resized, normalized, etc.).
- Model predicts the class label.
- API returns a JSON response:

```
{
"class": "Nike",
"confidence": [[0.89, 0.07, 0.04]]
}
```

Cloud Deployed API:

- The API is hosted on Google Cloud Run with a public endpoint.
- To test with an image using a Python script:

```
import requests
url =  https://nike-classifier-779080697580.us-central1.run.app/docs
files = {"file": open("test_shoe.jpg", "rb")}
response = requests.post(url, files=files)
print(response.json())
```

- Replace https://nike-classifier-779080697580.us-central1.run.app/docs with the actual deployed service URL.

How to Use the API:

1. Open Swagger UI for Interactive API Testing

Visit:

https://nike-classifier-779080697580.us-central1.run.app/docs

- Upload an image
- Click “Execute”
- View the classification result

2. Test API Using cURL:

Run the following command:

```
curl -X POST -F “file=@/path/to/image.jpg” \
https://nike-classifier-779080697580.us-central1.run.app
```

3. Use the API in a Python Script

```
import requests
url = “ https://nike-classifier-779080697580.us-central1.run.app “
files = {“file”: open(“/path/to/image.jpg”, “rb")}
response = requests.post(url, files=files)
print(response.json())  # {'class': 'Nike', 'confidence': [[0.89, 0.07, 0.04]]}
```

API Endpoints:

Endpoint	Method	Description
/	GET	Check API status
/docs	GET	Open Swagger UI
/predict	POST	Upload an image for classification

Example Request:

```
curl -X POST -F "file=@/path/to/image.jpg" \
   https://nike-classifier-xxxxxx.a.run.app/predict
```

Deployment Steps (Google Cloud Run):

- Set Google Cloud Project

```
gcloud config set project nikeidentifier
```

- Enable Required Services

```
gcloud services enable run.googleapis.com artifactregistry.googleapis.com
```

- Build and Push Docker Image

```
docker buildx build --platform linux/amd64 -t gcr.io/nikeidentifier/nike-classifier --push .
```

- Deploy to Cloud Run

```
gcloud run deploy nike-classifier \
          --image gcr.io/nikeidentifier/nike-classifier \
           --platform managed \
           --region us-central1 \
          --allow-unauthenticated \
          --port 8080 \
          --memory 2Gi
```

Security & Authentication:

Currently, the API is publicly accessible. Future enhancements include:

- API Key Authentication
- OAuth 2.0 Integration

- Google Cloud IAM Role-Based Access Control

Monitoring & Logging:

- Google Cloud Logging for request tracking
- Google Cloud Monitoring for uptime checks
- Error handling with structured logging

Future Enhancements:

- Improve model accuracy using more diverse training data
- Add support for real-time image processing via Cloud Storage
- Secure API with authentication & rate limiting
- Develop a mobile-friendly UI for uploading images

Summary:

- Nike Shoe Classifier is a serverless FastAPI service deployed on Google Cloud Run.
- The model classifies images into Nike, Other, and Non-Shoe categories.
- The API is fully dockerized, scalable, and supports automated deployments.
- Swagger UI (/docs) allows easy testing of the API.

Project Status: Fully Deployed & Ready for Use!