Face Mask Detection Project - Interview Script

Opening Statement (30-45 seconds)

"Thank you for the opportunity to discuss my Face Mask Detection project. This is a real-time computer vision application that uses deep learning to automatically detect whether a person is wearing a face mask through a webcam feed. It was particularly relevant during the COVID-19 pandemic and demonstrates practical applications of machine learning in healthcare and public safety settings."

Project Overview - In Simple Terms (1-2 minutes)

What the Project Does
"In simple terms, imagine you have a security camera at the entrance of a building. Instead of a security guard manually checking if everyone is wearing masks, this system does it automatically. When someone walks in front of the camera, the system looks at the video feed, analyzes the person's face, and instantly displays 'Mask' in green if they're wearing one, or 'No Mask' in red if they're not."

Why It Matters
"This is useful for places like hospitals, schools, airports, or any public space where mask compliance is important. It can help monitor compliance automatically, reduce manual oversight, and provide real-time feedback."

Technical Overview (2-3 minutes)

1. Problem Statement
"The challenge is teaching a computer to recognize the difference between someone wearing a mask and someone not wearing a mask. This is a binary classification problem in computer vision - meaning the model needs to distinguish between just two classes: 'with_mask' and 'without_mask'."

2. Dataset and Data Preparation
Simple Explanation:
"I collected a dataset of images - some showing people with masks and some without masks. The dataset is organized into training and testing folders, similar to how students learn from textbooks (training) and then take exams (testing)."

Technical Details:
- Dataset structure: Train/test split with two categories (with_mask, without_mask)

- Data preprocessing:
  - Images resized to 224x224 pixels (standard size for VGG16)
  - Normalization: Pixel values divided by 255 to scale between 0 and 1
  - Data shuffling to prevent bias
  - 80/20 train-test split using scikit-learn's `train_test_split`
- Final test accuracy: **94.9%** after optimization

3. Model Architecture - Transfer Learning with VGG16
Simple Explanation:
"Instead of building a model from scratch - which would be like learning to read without knowing the alphabet - I used something called 'transfer learning'. It's like taking a professional photographer's trained eye and adapting it to specifically recognize masks. I started with a pre-trained model called VGG16 that already knows how to recognize thousands of objects because it was trained on millions of images."

Technical Details:
- **Base Model**: VGG16 (Visual Geometry Group 16-layer model)
  - Pre-trained on ImageNet dataset (1.2 million images, 1000 classes)
  - 16 convolutional layers with max pooling
  - Proven architecture for image classification tasks

- **Transfer Learning Strategy**:
  - Removed the final classification layer (originally for 1000 ImageNet classes)
  - Froze all base layers (`layer.trainable = False`) to preserve learned features
  - Added a new dense layer with sigmoid activation for binary classification (with_mask/without_mask)
  - Only the final layer trains on our mask dataset

- **Why Transfer Learning?**
  - Reduces training time (5 epochs vs. potentially hundreds)
  - Requires less data (we don't need millions of mask images)
  - Leverages features learned from diverse images (edges, shapes, textures)

4. Model Training
Simple Explanation:
"I showed the model thousands of images repeatedly, telling it 'this person has a mask' or 'this person doesn't have a mask'. Each time it made a mistake, it adjusted itself slightly. After 5 rounds (epochs) through the data, it got really good at the task - achieving about 95% accuracy."

Technical Details:
- **Framework**: TensorFlow/Keras
- **Optimizer**: Adam (adaptive learning rate)
- **Loss Function**: Binary cross-entropy (standard for binary classification)

- **Metrics**: Accuracy
- **Training Process**:
  - 5 epochs (complete passes through training data)
  - Batch processing (32 images per batch)
  - Validation on 20% of data during training
  - Training time: ~150-170 seconds per epoch
- **Results**:
  - Final training accuracy: 94.06%
  - Final validation accuracy: 94.92%
  - Final test accuracy: 94.9%

5. Model Evaluation
Simple Explanation:
"To make sure the model actually works well and isn't just memorizing the training data, I tested it on images it had never seen before. I also used special metrics to understand exactly where it makes mistakes."

Technical Details:
- **Confusion Matrix Analysis**:
  - True Positives: 127 (correctly identified "no mask")
  - True Negatives: 116 (correctly identified "mask")
  - False Positives: 5 (predicted "no mask" when person had mask)
  - False Negatives: 8 (predicted "mask" when person didn't have mask)

- **Performance Metrics** (using 0.5 threshold):
  - Precision: 94% (when it says "mask", it's correct 94% of the time)
  - Recall: 96% (it catches 96% of all masked people)
  - F1-Score: 95% (balanced measure of precision and recall)

6. Real-Time Implementation
Simple Explanation:
"Finally, I connected the trained model to a webcam. Now it can analyze live video feed frame by frame - about 6-7 times per second - and display the result in real-time on the screen."

Technical Details:
- **Technology**: OpenCV (cv2) for video capture and image processing
- **Pipeline**:
  1. Capture frame from webcam using `cv2.VideoCapture(0)`
  2. Resize frame to 224x224 (model input requirement)
  3. Normalize pixel values (0-1 range)
  4. Pass through trained model for prediction
  5. Display result on frame:
     - Green label "Mask" if prediction = 0
     - Red label "No Mask" if prediction = 1

6. Real-time performance: ~150-175ms per frame (~6-7 FPS)

Technologies Used (30 seconds)

Libraries and Frameworks:
- **TensorFlow/Keras**: Deep learning framework for building and training the neural network
- **OpenCV (cv2)**: Computer vision library for image processing and video capture
- **NumPy**: Numerical operations and array manipulation
- **scikit-learn**: Data splitting and evaluation metrics
- **Python**: Programming language

Closing Statement (20 seconds)

"I'm proud of this project because it shows how machine learning can be applied to solve real-world problems efficiently. It combines computer vision, deep learning, and software engineering to create a working system that could actually be deployed in public spaces. I'd be happy to discuss any specific aspect in more detail or walk through the code."

Potential Follow-Up Questions & Answers

Q: Why VGG16 and not a newer model?
A: "VGG16 is a well-established, reliable architecture that's perfect for learning transfer learning concepts. It's also less complex than newer models like ResNet or EfficientNet, making it easier to understand and modify. For a production system, I would likely use ResNet50 or EfficientNetB0 for better accuracy-to-speed ratio."

Q: How did you handle data augmentation?
A: "I can see augmented images in the dataset (files prefixed with 'augmented_image_'), which suggests data augmentation was used to increase dataset diversity. This helps the model generalize better by seeing variations in lighting, angles, and backgrounds."

Q: What's the inference speed?
A: "The model processes approximately 6-7 frames per second (150-175ms per frame). While not super high-speed, it's sufficient for real-time monitoring applications. For faster inference, I could use model quantization or convert to TensorFlow Lite."