```
# CORRECTED CHEST X-RAY PNEUMONIA DETECTION MODEL
# Fixes: Prevents double-training overfitting


import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import os
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import numpy as np


# 1. DATA LOADING & PREPROCESSING



from google.colab import drive
drive.mount('/content/drive')

drive_path = '/content/drive/MyDrive/chest_xray'
IMAGE_SIZE = 224
BATCH_SIZE = 32
AUTOTUNE = tf.data.AUTOTUNE

# Load datasets
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    os.path.join(drive_path, 'train'),
    color_mode='grayscale',
    shuffle=True,
    image_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE
)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    os.path.join(drive_path, 'val'),
    color_mode='grayscale',
    shuffle=False,
    image_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE
)

test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    os.path.join(drive_path, 'test'),
    color_mode='grayscale',
    shuffle=False,
    image_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE
)

print(f"Train batches: {tf.data.experimental.cardinality(train_ds).numpy()}")
print(f"Val batches: {tf.data.experimental.cardinality(val_ds).numpy()}")
print(f"Test batches: {tf.data.experimental.cardinality(test_ds).numpy()}")


# 2. DATA AUGMENTATION (ONLY FOR TRAINING)


def grayscale_to_rgb(image, label):
    """Convert grayscale to RGB for ResNet50"""
    image = tf.image.grayscale_to_rgb(image)
    return image, label

def preprocess_fn(image, label):
    """Apply ResNet50 preprocessing"""
    image = preprocess_input(image)
    return image, label

# Data augmentation pipeline (only for training)
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
])

def augment_fn(image, label):
    """Apply augmentation with training flag"""
    image = data_augmentation(image, training=True)
    return image, label
```

```
    return image, label

# Convert to RGB first
train_ds = train_ds.map(grayscale_to_rgb, num_parallel_calls=AUTOTUNE)
val_ds = val_ds.map(grayscale_to_rgb, num_parallel_calls=AUTOTUNE)
test_ds = test_ds.map(grayscale_to_rgb, num_parallel_calls=AUTOTUNE)

# Apply augmentation ONLY to training data
train_ds = train_ds.map(augment_fn, num_parallel_calls=AUTOTUNE)
train_ds = train_ds.map(preprocess_fn, num_parallel_calls=AUTOTUNE)

# Preprocess validation & test (NO augmentation)
val_ds = val_ds.map(preprocess_fn, num_parallel_calls=AUTOTUNE)
test_ds = test_ds.map(preprocess_fn, num_parallel_calls=AUTOTUNE)

# Cache and prefetch for performance
train_ds = train_ds.cache().prefetch(AUTOTUNE)
val_ds = val_ds.cache().prefetch(AUTOTUNE)
test_ds = test_ds.cache().prefetch(AUTOTUNE)


# 3. BUILD MODEL WITH PROPER INITIALIZATION


# Load pre-trained ResNet50 (frozen base)
base_model = ResNet50(
    weights='imagenet',
    include_top=False,
    input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3)
)

# KEEP BASE FROZEN INITIALLY
base_model.trainable = False

# Build model architecture
inputs = tf.keras.Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3))
x = base_model(inputs, training=False)  # Use base in inference mode
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(128, activation='relu')(x)  # Additional dense layer
x = layers.Dropout(0.3)(x)  # Dropout to prevent overfitting
outputs = layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.Model(inputs, outputs)


# 4. FIRST TRAINING PHASE (FROZEN BASE ONLY)


print("\n" + "="*60)
print("PHASE 1: Training with frozen base (10 epochs max)")
print("="*60)

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

early_stop_phase1 = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True,
    verbose=1
)

history_phase1 = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=10,
    callbacks=[early_stop_phase1],
    verbose=1
)

# Evaluate after phase 1
print("\nPhase 1 - Test Set Evaluation:")
test_loss_p1, test_acc_p1 = model.evaluate(test_ds, verbose=0)
print(f"Test Loss: {test_loss_p1:.4f}, Test Accuracy: {test_acc_p1:.4f}")

# ============================================================
# 5. SECOND TRAINING PHASE (FINE-TUNING WITH LOWER LR)
# ============================================================

print("\n" + "="*60)
```

```
print("PHASE 2: Fine-tuning last 30 layers (5 epochs max)")
print("="*60)

# Unfreeze only the last 30 layers of the base model
base_model.trainable = True
for layer in base_model.layers[:-30]:  # Freeze all but last 30
    layer.trainable = False

# Use MUCH LOWER learning rate for fine-tuning
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Early stopping with patience
early_stop_phase2 = EarlyStopping(
    monitor='val_loss',
    patience=2,
    restore_best_weights=True,
    verbose=1
)

# Learning rate reduction
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=1,
    min_lr=1e-7,
    verbose=1
)

history_phase2 = model.fit(
    train_ds,  # SAME data, but with augmentation already applied
    validation_data=val_ds,
    epochs=5,
    callbacks=[early_stop_phase2, reduce_lr],
    verbose=1
)

# ============================================================
# 6. FINAL EVALUATION
# ============================================================

print("\n" + "="*60)
print("FINAL EVALUATION")
print("="*60)

test_loss, test_acc = model.evaluate(test_ds, verbose=0)
print(f"Final Test Loss: {test_loss:.4f}")
print(f"Final Test Accuracy: {test_acc:.4f}")

# ============================================================
# 7. REGULARIZATION TECHNIQUES APPLIED
# ============================================================

print("\n" + "="*60)
print("REGULARIZATION TECHNIQUES USED:")
print("="*60)
print("✓ Data Augmentation (Flip, Rotation, Zoom)")
print("✓ Dropout (0.3) in dense layer")
print("✓ Very low learning rate for fine-tuning (1e-5)")
print("✓ Early Stopping with patience=2")
print("✓ Learning Rate Reduction on plateau")
print("✓ Frozen base model layers (-30 layers kept frozen)")
print("✓ NO double-training on same data")

# ============================================================
# 8. SAVE MODEL
# ============================================================

drive_keras_path = '/content/drive/MyDrive/chest_xray_model_fixed.keras'
model.save(drive_keras_path)
print(f"\nModel saved to: {drive_keras_path}")

# ============================================================
# 9. VISUALIZATION OF TRAINING HISTORY
# ============================================================

import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 2, figsize=(12, 4))
```

```
# Phase 1
axes[0].plot(history_phase1.history['accuracy'], label='Train Acc', marker='o')
axes[0].plot(history_phase1.history['val_accuracy'], label='Val Acc', marker='s')
axes[0].set_title('Phase 1: Frozen Base Training')
axes[0].set_xlabel('Epoch')
axes[0].set_ylabel('Accuracy')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Phase 2
axes[1].plot(history_phase2.history['accuracy'], label='Train Acc', marker='o')
axes[1].plot(history_phase2.history['val_accuracy'], label='Val Acc', marker='s')
axes[1].set_title('Phase 2: Fine-tuning')
axes[1].set_xlabel('Epoch')
axes[1].set_ylabel('Accuracy')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('/content/drive/MyDrive/training_history.png', dpi=100, bbox_inches='tight')
print("Training history plot saved!")
plt.show()
```

```
Mounted at /content/drive
Found 5216 files belonging to 2 classes.
Found 16 files belonging to 2 classes.
Found 624 files belonging to 2 classes.
Train batches: 163
Val batches: 1
Test batches: 20
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_t
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step

============================================================
PHASE 1: Training with frozen base (10 epochs max)
============================================================
Epoch 1/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 1434s 9s/step - accuracy: 0.8798 - loss: 0.2735 - val_accuracy: 0.6875 - val_loss: 0.8880
Epoch 2/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 15s 90ms/step - accuracy: 0.9576 - loss: 0.1129 - val_accuracy: 0.6250 - val_loss: 0.7795
Epoch 3/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 14s 88ms/step - accuracy: 0.9652 - loss: 0.0895 - val_accuracy: 0.8125 - val_loss: 0.5401
Epoch 4/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 14s 85ms/step - accuracy: 0.9671 - loss: 0.0748 - val_accuracy: 0.8750 - val_loss: 0.2508
Epoch 5/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 14s 83ms/step - accuracy: 0.9724 - loss: 0.0736 - val_accuracy: 0.8750 - val_loss: 0.3126
Epoch 6/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 14s 84ms/step - accuracy: 0.9781 - loss: 0.0543 - val_accuracy: 0.9375 - val_loss: 0.1063
Epoch 7/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 14s 84ms/step - accuracy: 0.9776 - loss: 0.0582 - val_accuracy: 0.8750 - val_loss: 0.1230
Epoch 8/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 14s 86ms/step - accuracy: 0.9789 - loss: 0.0550 - val_accuracy: 1.0000 - val_loss: 0.0701
Epoch 9/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 14s 85ms/step - accuracy: 0.9823 - loss: 0.0472 - val_accuracy: 0.8750 - val_loss: 0.1585
Epoch 10/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 20s 84ms/step - accuracy: 0.9863 - loss: 0.0369 - val_accuracy: 0.8750 - val_loss: 0.1412
Restoring model weights from the end of the best epoch: 8.

Phase 1 - Test Set Evaluation:
Test Loss: 0.3734, Test Accuracy: 0.8862

============================================================
PHASE 2: Fine-tuning last 30 layers (5 epochs max)
============================================================
Epoch 1/5
163/163 ━━━━━━━━━━━━━━━━━━━━ 46s 157ms/step - accuracy: 0.9598 - loss: 0.1135 - val_accuracy: 0.9375 - val_loss: 0.1607 - le
Epoch 2/5
163/163 ━━━━━━━━━━━━━━━━━━━━ 0s 120ms/step - accuracy: 0.9979 - loss: 0.0145
Epoch 2: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-06.
163/163 ━━━━━━━━━━━━━━━━━━━━ 20s 120ms/step - accuracy: 0.9979 - loss: 0.0145 - val_accuracy: 0.9375 - val_loss: 0.2494 - le
Epoch 3/5
163/163 ━━━━━━━━━━━━━━━━━━━━ 0s 120ms/step - accuracy: 0.9997 - loss: 0.0052
Epoch 3: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-06.
163/163 ━━━━━━━━━━━━━━━━━━━━ 20s 120ms/step - accuracy: 0.9997 - loss: 0.0052 - val_accuracy: 0.9375 - val_loss: 0.2634 - le
Epoch 3: early stopping
Restoring model weights from the end of the best epoch: 1.

============================================================
FINAL EVALUATION
```