In [4]:

```
#!/usr/bin/env python

import os
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.callbacks import Callback

base_dir = "./"
train_dir = os.path.join(base_dir, "train")
test_dir = os.path.join(base_dir, "test")

# Data augmentation and data generators
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(112, 112),
    batch_size=10,
    class_mode='binary' # Using binary classification
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(112, 112),
    batch_size=10,
    class_mode='binary' # Using binary classification
)

# Network based on VGG19
conv_base = VGG19(include_top=False, input_shape=(112, 112, 3), weights='imagenet')
for layer in conv_base.layers:
    layer.trainable = False

z = conv_base.output
z = GlobalAveragePooling2D()(z)
z = Dense(128, activation='relu')(z)
z = Dropout(0.5)(z)
predictions = Dense(1, activation='sigmoid')(z)

model = Model(inputs=conv_base.input, outputs=predictions)
```

```
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])

# Logging to text file
log_filename = "training_log.txt"

class LogCallback(Callback):
    def __init__(self, log_filename):
        super().__init__()
        self.log_filename = log_filename
        self.log_file = open(self.log_filename, 'w')

    def on_epoch_end(self, epoch, logs=None):
        log_message = f"Epoch {epoch + 1}, Loss: {logs['loss']}, Accuracy: {logs['accuracy']}, Val Loss: {logs['val_loss']}, Val Accuracy: {logs['val_accuracy']}\n"
        print(log_message, end='')
        self.log_file.write(log_message)

    def on_train_end(self, logs=None):
        self.log_file.close()

    def print_final_results(self, history):
        # Assuming history is a dictionary containing 'accuracy', 'val_accuracy', 'loss', 'val_loss'
        final_acc = history['accuracy'][-1]
        final_val_acc = history['val_accuracy'][-1]
        final_loss = history['loss'][-1]
        final_val_loss = history['val_loss'][-1]
        print(f"\nFinal Training Accuracy: {final_acc}, Final Validation Accuracy: {final_val_acc}")
        print(f"Final Training Loss: {final_loss}, Final Validation Loss: {final_val_loss}\n")

history = model.fit(
    train_generator,
    epochs=20,
    validation_data=test_generator,
    callbacks=[LogCallback(log_filename)], # Ensure LogCallback uses the file properly
    verbose=1
)

log_callback = LogCallback(log_filename)
log_callback.print_final_results(history.history)

model.save("vgg19_chihuahua_vs_muffin.keras")

# Getting data from training history
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

# Saving graphs to PDF
with PdfPages('training_results.pdf') as pdf:
    fig, ax = plt.subplots(1, 2, figsize=(12, 5))

    ax[0].plot(epochs, acc, 'b', label='Training accuracy')
    ax[0].plot(epochs, val_acc, 'r', label='Validation accuracy')
    ax[0].set_title('Training and Validation Accuracy')
    ax[0].legend()
```

```
    ax[1].plot(epochs, loss, 'b', label='Training Loss')
    ax[1].plot(epochs, val_loss, 'r', label='Validation Loss')
    ax[1].set_title('Training and Validation Loss')
    ax[1].legend()

    pdf.savefig(fig)
    plt.close(fig)

    plt.show()

    pdf.savefig(fig)
    plt.close(fig)
```

```
Found 4733 images belonging to 2 classes.
Found 1184 images belonging to 2 classes.
Epoch 1/20
474/474 [==============================] - ETA: 0s - loss: 0.3280 - accuracy: 0.8565Epoch 1, Loss: 0.32804179191589355, Accuracy: 0.8565391898155212, Val Loss: 0.1817649
1558551788, Val Accuracy: 0.9341216087341309
474/474 [==============================] - 75s 157ms/step - loss: 0.3280 - accuracy: 0.8565 - val_loss: 0.1818 - val_accuracy: 0.9341
Epoch 2/20
474/474 [==============================] - ETA: 0s - loss: 0.2434 - accuracy: 0.8977Epoch 2, Loss: 0.2433685064315796, Accuracy: 0.8977392911911011, Val Loss: 0.17329645
156860352, Val Accuracy: 0.9383445978164673
474/474 [==============================] - 75s 157ms/step - loss: 0.2434 - accuracy: 0.8977 - val_loss: 0.1733 - val_accuracy: 0.9383
Epoch 3/20
474/474 [==============================] - ETA: 0s - loss: 0.2333 - accuracy: 0.9062Epoch 3, Loss: 0.23334358632564545, Accuracy: 0.9061905741691589, Val Loss: 0.1650282
2935581207, Val Accuracy: 0.9417229890823364
474/474 [==============================] - 74s 155ms/step - loss: 0.2333 - accuracy: 0.9062 - val_loss: 0.1650 - val_accuracy: 0.9417
Epoch 4/20
474/474 [==============================] - ETA: 0s - loss: 0.2251 - accuracy: 0.9013Epoch 4, Loss: 0.2250588983297348, Accuracy: 0.9013310670852661, Val Loss: 0.15402284
264564514, Val Accuracy: 0.943412184715271
474/474 [==============================] - 74s 155ms/step - loss: 0.2251 - accuracy: 0.9013 - val_loss: 0.1540 - val_accuracy: 0.9434
Epoch 5/20
474/474 [==============================] - ETA: 0s - loss: 0.2102 - accuracy: 0.9123Epoch 5, Loss: 0.21016967296600342, Accuracy: 0.9123177528381348, Val Loss: 0.1510116
7559623718, Val Accuracy: 0.943412184715271
474/474 [==============================] - 74s 155ms/step - loss: 0.2102 - accuracy: 0.9123 - val_loss: 0.1510 - val_accuracy: 0.9434
Epoch 6/20
474/474 [==============================] - ETA: 0s - loss: 0.2097 - accuracy: 0.9125Epoch 6, Loss: 0.20969770848751068, Accuracy: 0.9125290513038635, Val Loss: 0.1577412
188053131, Val Accuracy: 0.9408783912658691
474/474 [==============================] - 75s 159ms/step - loss: 0.2097 - accuracy: 0.9125 - val_loss: 0.1577 - val_accuracy: 0.9409
Epoch 7/20
474/474 [==============================] - ETA: 0s - loss: 0.2117 - accuracy: 0.9151Epoch 7, Loss: 0.21174271404743195, Accuracy: 0.9150644540786743, Val Loss: 0.1511732
6378822327, Val Accuracy: 0.9349662065505981
474/474 [==============================] - 73s 155ms/step - loss: 0.2117 - accuracy: 0.9151 - val_loss: 0.1512 - val_accuracy: 0.9350
Epoch 8/20
474/474 [==============================] - ETA: 0s - loss: 0.2000 - accuracy: 0.9155Epoch 8, Loss: 0.19996798038482666, Accuracy: 0.9154869914054871, Val Loss: 0.1495032
7575206757, Val Accuracy: 0.9425675868988037
474/474 [==============================] - 74s 156ms/step - loss: 0.2000 - accuracy: 0.9155 - val_loss: 0.1495 - val_accuracy: 0.9426
Epoch 9/20
474/474 [==============================] - ETA: 0s - loss: 0.2010 - accuracy: 0.9144Epoch 9, Loss: 0.2010054886341095, Accuracy: 0.9144306182861328, Val Loss: 0.14661842
584609985, Val Accuracy: 0.9417229890823364
474/474 [==============================] - 74s 157ms/step - loss: 0.2010 - accuracy: 0.9144 - val_loss: 0.1466 - val_accuracy: 0.9417
Epoch 10/20
474/474 [==============================] - ETA: 0s - loss: 0.1837 - accuracy: 0.9222Epoch 10, Loss: 0.18371383845806122, Accuracy: 0.9222480654716492, Val Loss: 0.146809
1905117035, Val Accuracy: 0.9408783912658691
474/474 [==============================] - 74s 156ms/step - loss: 0.1837 - accuracy: 0.9222 - val_loss: 0.1468 - val_accuracy: 0.9409
Epoch 11/20
474/474 [==============================] - ETA: 0s - loss: 0.1877 - accuracy: 0.9258Epoch 11, Loss: 0.1876940280199051, Accuracy: 0.9258398413658142, Val Loss: 0.1439255
```

```
177974701, Val Accuracy: 0.9417229890823364
474/474 [==============================] - 75s 157ms/step - loss: 0.1877 - accuracy: 0.9258 - val_loss: 0.1439 - val_accuracy: 0.9417
Epoch 12/20
474/474 [==============================] - ETA: 0s - loss: 0.1920 - accuracy: 0.9210Epoch 12, Loss: 0.19203895330429077, Accuracy: 0.9209803342819214, Val Loss: 0.143456
3547372818, Val Accuracy: 0.9476351141929626
474/474 [==============================] - 74s 156ms/step - loss: 0.1920 - accuracy: 0.9210 - val_loss: 0.1435 - val_accuracy: 0.9476
Epoch 13/20
474/474 [==============================] - ETA: 0s - loss: 0.1867 - accuracy: 0.9241Epoch 13, Loss: 0.1866559088230133, Accuracy: 0.9241495728492737, Val Loss: 0.1389803
2903671265, Val Accuracy: 0.9425675868988037
474/474 [==============================] - 75s 159ms/step - loss: 0.1867 - accuracy: 0.9241 - val_loss: 0.1390 - val_accuracy: 0.9426
Epoch 14/20
474/474 [==============================] - ETA: 0s - loss: 0.1813 - accuracy: 0.9267Epoch 14, Loss: 0.18128317594528198, Accuracy: 0.9266849756240845, Val Loss: 0.167845
6813097, Val Accuracy: 0.931587815284729
474/474 [==============================] - 74s 157ms/step - loss: 0.1813 - accuracy: 0.9267 - val_loss: 0.1678 - val_accuracy: 0.9316
Epoch 15/20
474/474 [==============================] - ETA: 0s - loss: 0.1850 - accuracy: 0.9254Epoch 15, Loss: 0.18496687710285187, Accuracy: 0.9254173040390015, Val Loss: 0.154361
3076210022, Val Accuracy: 0.9332770109176636
474/474 [==============================] - 75s 157ms/step - loss: 0.1850 - accuracy: 0.9254 - val_loss: 0.1544 - val_accuracy: 0.9333
Epoch 16/20
474/474 [==============================] - ETA: 0s - loss: 0.1802 - accuracy: 0.9280Epoch 16, Loss: 0.1801726222038269, Accuracy: 0.9279526472091675, Val Loss: 0.1382881
2539577484, Val Accuracy: 0.9425675868988037
474/474 [==============================] - 74s 157ms/step - loss: 0.1802 - accuracy: 0.9280 - val_loss: 0.1383 - val_accuracy: 0.9426
Epoch 17/20
474/474 [==============================] - ETA: 0s - loss: 0.1794 - accuracy: 0.9282Epoch 17, Loss: 0.17935600876808167, Accuracy: 0.9281639456748962, Val Loss: 0.170915
6185388565, Val Accuracy: 0.9298986196517944
474/474 [==============================] - 75s 159ms/step - loss: 0.1794 - accuracy: 0.9282 - val_loss: 0.1709 - val_accuracy: 0.9299
Epoch 18/20
474/474 [==============================] - ETA: 0s - loss: 0.1721 - accuracy: 0.9299Epoch 18, Loss: 0.1720798909664154, Accuracy: 0.9298542141914368, Val Loss: 0.1326835
3044986725, Val Accuracy: 0.9501689076423645
474/474 [==============================] - 74s 157ms/step - loss: 0.1721 - accuracy: 0.9299 - val_loss: 0.1327 - val_accuracy: 0.9502
Epoch 19/20
474/474 [==============================] - ETA: 0s - loss: 0.1793 - accuracy: 0.9256Epoch 19, Loss: 0.17925158143043518, Accuracy: 0.9256285429000854, Val Loss: 0.136047
825217247, Val Accuracy: 0.9442567825317383
474/474 [==============================] - 75s 159ms/step - loss: 0.1793 - accuracy: 0.9256 - val_loss: 0.1360 - val_accuracy: 0.9443
Epoch 20/20
474/474 [==============================] - ETA: 0s - loss: 0.1725 - accuracy: 0.9303Epoch 20, Loss: 0.1725095957517624, Accuracy: 0.9302767515182495, Val Loss: 0.1360098
8686084747, Val Accuracy: 0.9442567825317383
474/474 [==============================] - 76s 160ms/step - loss: 0.1725 - accuracy: 0.9303 - val_loss: 0.1360 - val_accuracy: 0.9443

Final Training Accuracy: 0.9302767515182495, Final Validation Accuracy: 0.9442567825317383
Final Training Loss: 0.1725095957517624, Final Validation Loss: 0.13600988686084747
```

In [ ]:

```
На основе предоставленных логов обучения, используемая архитектура нейронной сети основана на модели VGG19 с некоторыми модификациями,
включая добавление слоя GlobalAveragePooling2D, слоя Dense с 128 нейронами и функцией активации ReLU,
слоя Dropout с коэффициентом 0.5 и финального слоя Dense с сигмоидной функцией активации для бинарной классификации.
Эта архитектура является хорошим выбором для задач классификации изображений, особенно при работе с большим количеством классов,
так как она сочетает в себе преимущества сверточных нейронных сетей (CNN) для извлечения признаков и полностью связанных слоев для классификации.

Лог обучения (training_log.txt) показывают постепенное улучшение точности как на обучающем, так и на валидационном наборах данных,
с достижением валидационной точности 94,43% к концу 20-й эпохи.
 Это указывает на то, что модель хорошо обобщает данные, что является хорошим признаком ее производительности.

Использованные методы аугментации данных, такие как вращение, сдвиги по ширине и высоте, сдвиг, увеличение
и уменьшение масштаба, а также горизонтальное отражение, являются критически важными для улучшения способности модели обобщать данные.
Эти техники создают вариации обучающих изображений, помогая модели изучать более устойчивые признаки и снижая переобучение.
Нормализация изображений путем деления на 255 также является стандартным шагом предварительной обработки,
```

который помогает нормализовать входные данные, что положительно влияет на процесс обучения.

Использование dropout во время обучения, в частности с коэффициентом 0.5, является формой регуляризации,
которая помогает предотвратить переобучение, случайно устанавливая долю входных единиц в 0 на каждом обновлении во время обучения.
Этот метод заставляет модель изучать более устойчивые признаки и улучшает способность модели обобщать данные.

Выбор оптимизатора Adam также имеет значение. Adam - это адаптивный алгоритм оптимизации скорости обучения,
который хорошо подходит для обучения глубоких нейронных сетей. Он сочетает в себе преимущества двух других
расширений стохастического градиентного спуска: AdaGrad и RMSProp. Adam известен своей эффективностью и
эффективностью в обработке разреженных градиентов на шумных задачах, что полезно для задач классификации изображений.

В заключение, комбинация архитектуры VGG19 с аугментацией данных, регуляризацией dropout и оптимизатором Adam привела к модели,
достигшей высокой точности как на обучающем, так и на валидационном наборах данных. Аугментация данных и регуляризация dropout
были особенно эффективны в улучшении производительности модели, как это подтверждается постепенным улучшением точности на протяжении
всего процесса обучения.