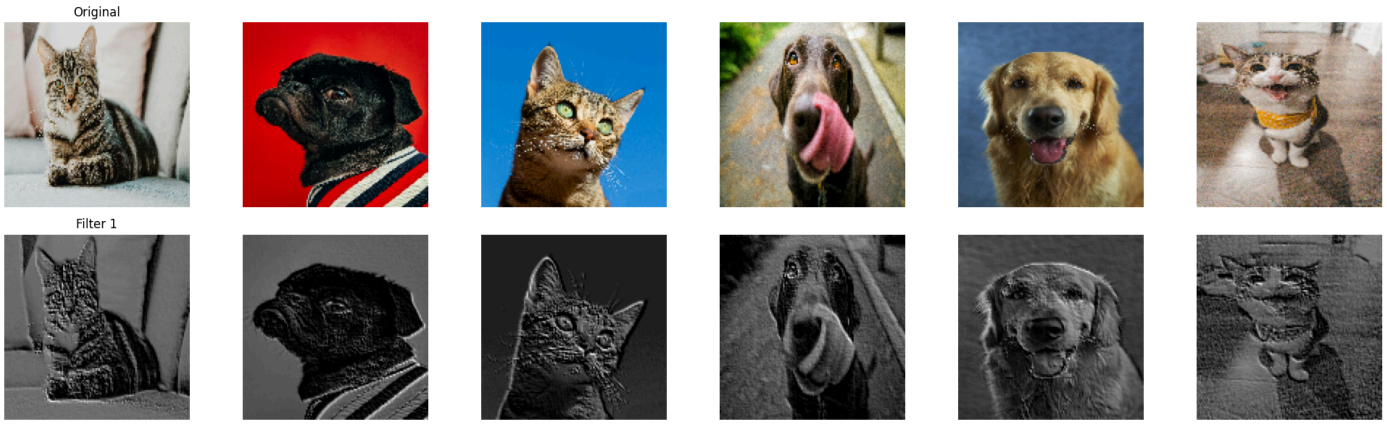
```
# Step 1: Install & Import Required Libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import requests
from PIL import Image
from io import BytesIO

# Step 2: Create directory and list of image URLs
os.makedirs("data/cats_dogs", exist_ok=True)

image_urls = [
    # Cats
    "https://images.unsplash.com/photo-1518791841217-8f162f1e1131",
    "https://images.unsplash.com/photo-1603318050250-84f50bbdcbdc",
    "https://images.unsplash.com/photo-1517423440428-a5a00ad493e8",
    "https://images.unsplash.com/photo-1574158622682-e40e69881006",
    "https://images.unsplash.com/photo-1560114928-40f1f1eb26a9",
    # Dogs
    "https://images.unsplash.com/photo-1601758123927-1961f9c88e52",
    "https://images.unsplash.com/photo-1518717758536-85ae29035b6d",
    "https://images.unsplash.com/photo-1558788353-f76d92427f16",
    "https://images.unsplash.com/photo-1543852786-1cf6624b9987",
    "https://images.unsplash.com/photo-1583337130417-3346a1a4a4c0",
]

# Step 3: Download and validate images
image_paths = []
for i, url in enumerate(image_urls):
    try:
        response = requests.get(url, timeout=10)
        img = Image.open(BytesIO(response.content)).convert("RGB")
        path = f"data/cats_dogs/image_{i}.jpg"
        img.save(path)
        image_paths.append(path)
    except Exception as e:
        print(f"Failed to download image {i}: {e}")

# Step 4: Load & Preprocess Images
img_size = (128, 128)
images = []
for path in image_paths:
    try:
        img = load_img(path, target_size=img_size)
        img_array = img_to_array(img) / 255.0
        images.append(img_array)
    except Exception as e:
        print(f"Failed to process image {path}: {e}")

images = np.array(images)

# Step 5: Define a Simple Conv2D Model
model = Sequential([
    Conv2D(filters=8, kernel_size=(3, 3), activation='relu', input_shape=(128, 128, 3))
])

# Step 6: Apply the Convolutional Layer
conv_output = model.predict(images)

# Step 7: Visualize Filtered Output
def plot_feature_maps(originals, filtered, filter_index=0):
    plt.figure(figsize=(20, 6))
    for i in range(len(originals)):
        plt.subplot(2, len(originals), i + 1)
        plt.imshow(originals[i])
        plt.axis('off')
        if i == 0:
            plt.title("Original")

        plt.subplot(2, len(originals), len(originals) + i + 1)
        plt.imshow(filtered[i, :, :, filter_index], cmap='gray')
        plt.axis('off')
```

```
        if i == 0:
            plt.title(f"Filter {filter_index+1}")
    plt.tight_layout()
    plt.show()

plot_feature_maps(images, conv_output, filter_index=0)
```

```
Failed to download image 1: cannot identify image file <_io.BytesIO object at 0x7d7320fbc950>
Failed to download image 4: cannot identify image file <_io.BytesIO object at 0x7d7320fbccc0>
Failed to download image 5: cannot identify image file <_io.BytesIO object at 0x7d7320fbcef0>
Failed to download image 9: cannot identify image file <_io.BytesIO object at 0x7d7320fbffb0>
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 212ms/step
```




```
# Step 1: Import Required Libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import requests
from PIL import Image
from io import BytesIO

# Step 2: Download and Save Images (5 cats, 5 dogs)
os.makedirs("data/cats_dogs", exist_ok=True)
image_urls = [
    # Cats
    "https://images.unsplash.com/photo-1518791841217-8f162f1e1131",
    "https://images.unsplash.com/photo-1603318050250-84f50bbdcbdc",
    "https://images.unsplash.com/photo-1517423440428-a5a00ad493e8",
    "https://images.unsplash.com/photo-1574158622682-e40e69881006",
    "https://images.unsplash.com/photo-1560114928-40f1f1eb26a9",
    # Dogs
    "https://images.unsplash.com/photo-1601758123927-1961f9c88e52",
    "https://images.unsplash.com/photo-1518717758536-85ae29035b6d",
    "https://images.unsplash.com/photo-1558788353-f76d92427f16",
    "https://images.unsplash.com/photo-1543852786-1cf6624b9987",
    "https://images.unsplash.com/photo-1583337130417-3346a1a4a4c0",
]

labels = [0]*5 + [1]*5  # 0 = cat, 1 = dog

image_paths = []
for i, url in enumerate(image_urls):
    try:
        response = requests.get(url, timeout=10)
        # Check if the request was successful (status code 200)
        if response.status_code == 200:
            img = Image.open(BytesIO(response.content)).convert("RGB")
            path = f"data/cats_dogs/image_{i}.jpg"
```

```
            img.save(path)
            image_paths.append(path)
        else:
            print(f"Failed to download image {i}: Status code {response.status_code}")
    except Exception as e:
        print(f"Failed to download image {i}: {e}")

# Step 3: Load & Preprocess Images
img_size = (128, 128)
images = []
# Only load images and labels for successfully downloaded images
for i, path in enumerate(image_paths):
    try:
        img = load_img(path, target_size=img_size)
        img_array = img_to_array(img) / 255.0
        images.append(img_array)
    except Exception as e:
        print(f"Failed to process image {path}: {e}")

X = np.array(images)
# Adjust labels to match the number of loaded images
y = np.array(labels[:len(image_paths)]) # Use slicing here to ensure labels match images

# Step 4: Build CNN + FFN Model
model = Sequential([
    Conv2D(filters=16, kernel_size=(3, 3), activation='relu', input_shape=(128, 128, 3)),
    Flatten(),  # Converts feature maps to 1D vector
    Dense(64, activation='relu'),  # FFN layer
    Dense(1, activation='sigmoid')  # Output layer for binary classification
])

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

# Step 5: Train the Model
model.fit(X, y, epochs=10, batch_size=2, verbose=1)

# Step 6: Evaluate
loss, acc = model.evaluate(X, y, verbose=0)
print(f"\n✅ Training Accuracy: {acc*100:.2f}%")

# Optional: Predict
preds = model.predict(X)
print("\nPredictions:", preds.round().flatten())
```

```
Failed to download image 1: Status code 404
Failed to download image 4: Status code 404
Failed to download image 5: Status code 404
Failed to download image 9: Status code 404
Epoch 1/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 377ms/step - accuracy: 0.3750 - loss: 15.1509
Epoch 2/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 429ms/step - accuracy: 0.9167 - loss: 5.1884
Epoch 3/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 266ms/step - accuracy: 0.4375 - loss: 7.0261
Epoch 4/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 273ms/step - accuracy: 0.5000 - loss: 8.8943
Epoch 5/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 266ms/step - accuracy: 1.0000 - loss: 2.0273e-04
Epoch 6/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 273ms/step - accuracy: 0.9167 - loss: 2.1458
Epoch 7/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 266ms/step - accuracy: 0.9167 - loss: 1.7729
Epoch 8/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 272ms/step - accuracy: 0.8542 - loss: 1.1293
Epoch 9/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 268ms/step - accuracy: 1.0000 - loss: 7.0455e-07
Epoch 10/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 267ms/step - accuracy: 1.0000 - loss: 1.0516e-08

✅ Training Accuracy: 100.00%
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step

Predictions: [0. 0. 0. 0. 0. 1.]
```

```
import os
import numpy as np
import requests
import matplotlib.pyplot as plt
from PIL import Image
```

```
from io import BytesIO
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, Dense, Reshape, Input
from tensorflow.keras.preprocessing.image import load_img, img_to_array

# Download & Preprocess 10 Images (5 cats + 5 dogs)
os.makedirs("data/cats_dogs", exist_ok=True)

urls = [
    # Cats
    "https://images.unsplash.com/photo-1518791841217-8f162f1e1131",
    "https://images.unsplash.com/photo-1603318050250-84f50bbdcbdc",
    "https://images.unsplash.com/photo-1517423440428-a5a00ad493e8",
    "https://images.unsplash.com/photo-1574158622682-e40e69881006",
    "https://images.unsplash.com/photo-1560114928-40f1f1eb26a9",
    # Dogs
    "https://images.unsplash.com/photo-1601758123927-1961f9c88e52",
    "https://images.unsplash.com/photo-1518717758536-85ae29035b6d",
    "https://images.unsplash.com/photo-1558788353-f76d92427f16",
    "https://images.unsplash.com/photo-1543852786-1cf6624b9987",
    "https://images.unsplash.com/photo-1583337130417-3346a1a4a4c0",
]
labels = [0]*5 + [1]*5  # 0=cat, 1=dog

# Load images and match with labels
images = []
final_labels = []
img_size = (64, 64)
for i, (url, label) in enumerate(zip(urls, labels)):
    try:
        response = requests.get(url, timeout=10)
        img = Image.open(BytesIO(response.content)).convert("RGB")
        img = img.resize(img_size)
        img_array = img_to_array(img) / 255.0
        images.append(img_array)
        final_labels.append(label)
    except:
        print(f"Skipping image {i}")

X = np.array(images)
y = np.array(final_labels)

print("✅ Loaded images:", X.shape)

# Build CNN → FFN → Reshape → CNN model
model = Sequential([
    Input(shape=(64, 64, 3)),

    # 1st CNN
    Conv2D(16, (3, 3), activation='relu', padding='same'),

    # Flatten to go to FFN
    Flatten(),

    # FFN
    Dense(1024, activation='relu'),

    # Reshape FFN output back to "image" (for 2nd CNN)
    Reshape((16, 16, 4)),  # You control this shape so long as 16*16*4 = 1024

    # 2nd CNN after FFN
    Conv2D(8, (3, 3), activation='relu', padding='same'),

    # Final Classification
    Flatten(),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()

# Train the model
model.fit(X, y, epochs=10, batch_size=2, verbose=1)

# Evaluate
loss, acc = model.evaluate(X, y, verbose=0)
```

```
print(f"\n🎯 Final Accuracy on 10 images: {acc*100:.2f}%")
```

```
Skipping image 1
Skipping image 4
Skipping image 5
Skipping image 9
✅ Loaded images: (6, 64, 64, 3)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 16)	448
flatten (Flatten)	(None, 65536)	0
dense (Dense)	(None, 1024)	67,109,888
reshape (Reshape)	(None, 16, 16, 4)	0
conv2d_1 (Conv2D)	(None, 16, 16, 8)	296
flatten_1 (Flatten)	(None, 2048)	0
dense_1 (Dense)	(None, 1)	2,049

```
 Total params: 67,112,681 (256.01 MB)
 Trainable params: 67,112,681 (256.01 MB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.2708 - loss: 10.0595
Epoch 2/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 5s 2s/step - accuracy: 0.3542 - loss: 11.0620
Epoch 3/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 5s 2s/step - accuracy: 0.8542 - loss: 2.0928
Epoch 4/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 0.5625 - loss: 1.1100
Epoch 5/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.7292 - loss: 0.4345
Epoch 6/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.8542 - loss: 0.4375
Epoch 7/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 1.0000 - loss: 5.1271e-05
Epoch 8/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 1.0000 - loss: 1.9228e-05
Epoch 9/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 1.0000 - loss: 4.4659e-05
Epoch 10/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 1.0000 - loss: 1.6783e-04

🎯 Final Accuracy on 10 images: 100.00%
```