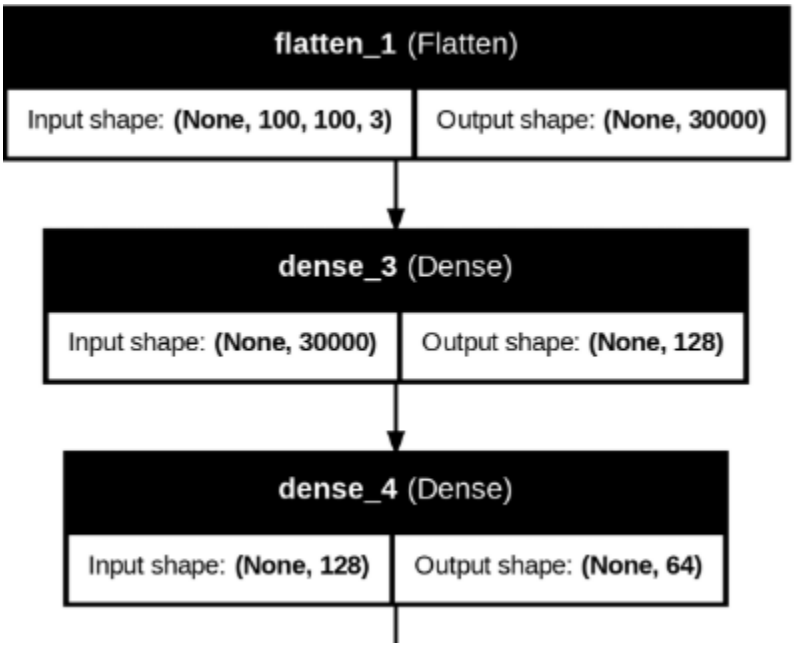
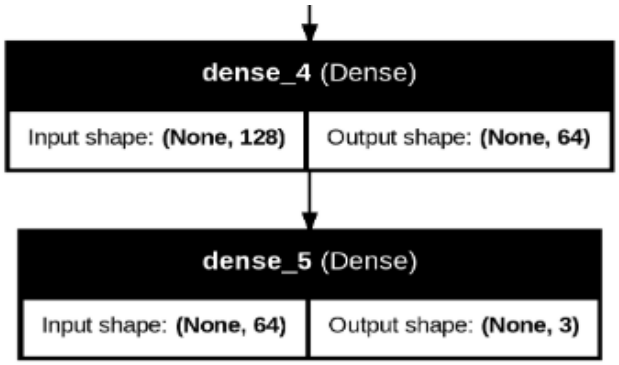
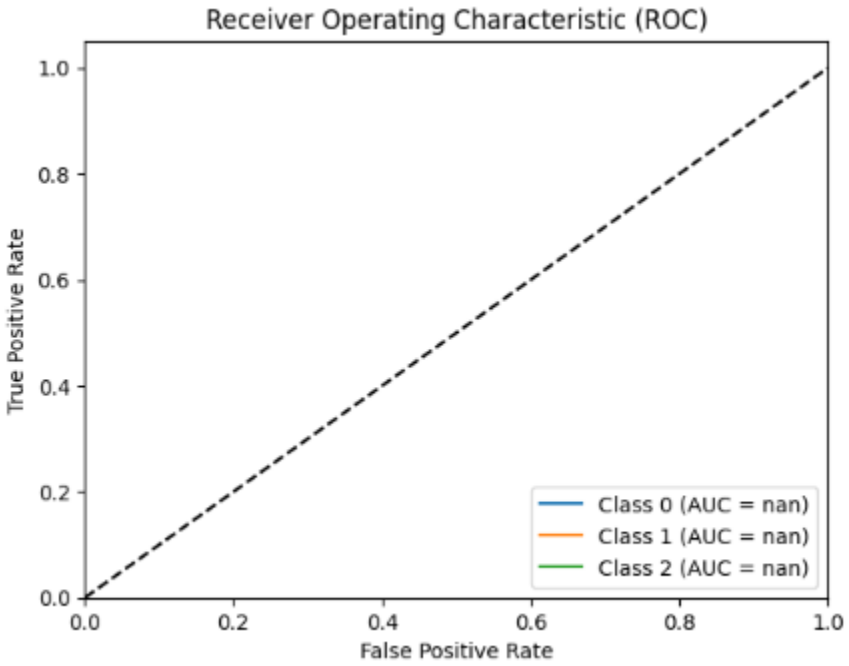
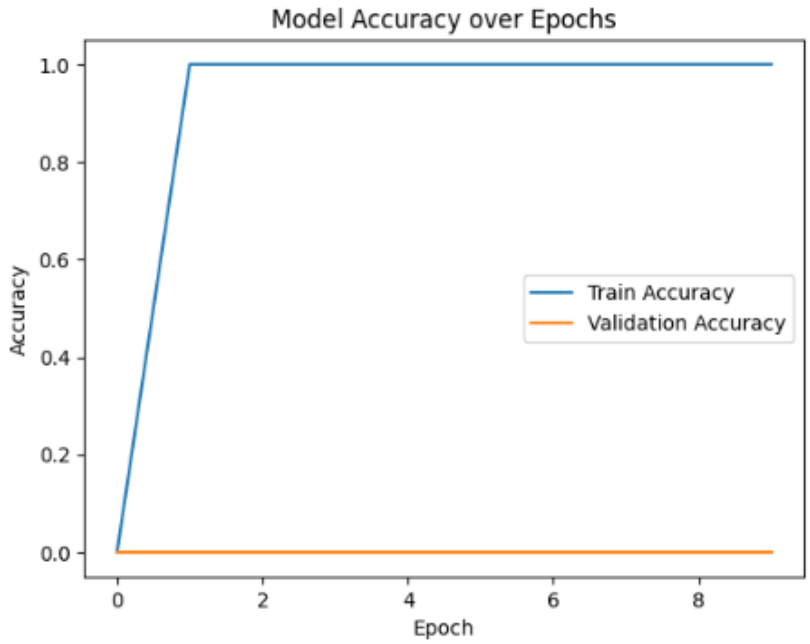
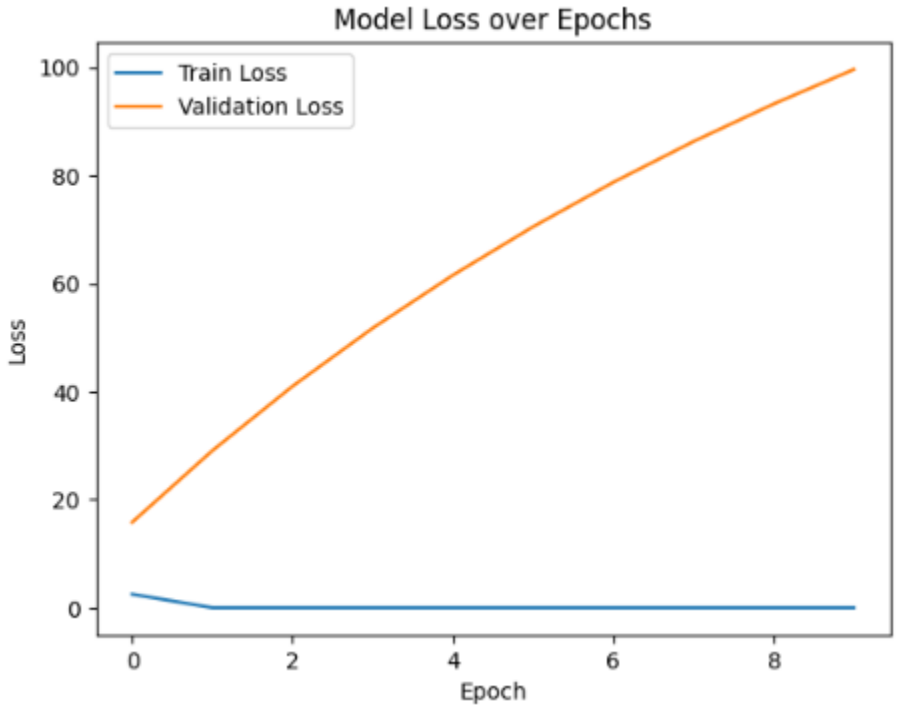
```
Error loading image: 20.jpg
(2, 100, 100, 3) (2, 3)
Epoch 1/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 1.0000 - loss: 0.3462
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 141ms/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 138ms/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 9/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 142ms/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 10/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 1.0000 - loss: 0.0000e+00
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 221ms/step - accuracy: 0.0000e+00 - loss: 295.6022
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
Test Accuracy: 0.0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step
The predicted currency denomination is: 10
```




dense_4 (Dense)
Input shape: (None, 128)
Output shape: (None, 64)
dense_5 (Dense)
Input shape: (None, 64)
Output shape: (None, 3)


Receiver Operating Characteristic (ROC)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 0 (AUC = nan)
Class 1 (AUC = nan)
Class 2 (AUC = nan)






```
import cv2
import numpy as np
import os
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
import matplotlib.pyplot as plt
```

```python
import os
import cv2
import numpy as np

# List uploaded files in the current directory to verify if they are
correctly uploaded
uploaded_files = os.listdir()
print("Uploaded files:", uploaded_files)

# Mapping each image to its label
label_map = {
    "10.jpg": 0,
    "20.jpg": 1,
    "100.jpg": 2
}

data = []
labels = []

# Loop through each image and load them
for filename, label in label_map.items():
    if filename in uploaded_files:
        # Read image in color mode (BGR)
        img = cv2.imread(filename)
        if img is not None:
            # Resize to 100x100 for consistency
            img = cv2.resize(img, (100, 100))
            # Normalize the image (scale pixel values to [0, 1])
            img = img / 255.0
            data.append(img)
            labels.append(label)
        else:
            print(f"Error loading image: {filename}")
    else:
        print(f"Image {filename} not found in the directory!")

# Convert to numpy arrays
X = np.array(data)
y = np.array(labels)

# Check the shape of the data and labels
print(X.shape, y.shape)
```

```
import cv2
import numpy as np
import os
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical

# Load and process the images
# Update with your image paths

# Set the path where your images are stored
image_dir = "/content"  # Update this if using Colab (Google Drive or
local path)

# Define the label map and the associated image files
label_map = {
    "10.jpg": 0,
    "20.jpg": 1,
    "100.jpg": 2
}

data = []
labels = []

# Loop through each image and load them
for filename, label in label_map.items():
    img_path = os.path.join(image_dir, filename)

    # Read image in color mode (BGR)
    img = cv2.imread(img_path)

    if img is not None:
        # Resize to 100x100 for consistency
        img = cv2.resize(img, (100, 100))

        # Normalize the image (scale pixel values to [0, 1])
        img = img / 255.0

        # Append the image data and corresponding label
        data.append(img)
        labels.append(label)
    else:
        print(f"Error loading image: {filename}")
```

```
# Convert lists to numpy arrays
X = np.array(data)
y = np.array(labels)

# One-hot encode the labels (as we have 3 classes)
y = to_categorical(y, 3)

# Check the shape of the data and labels
print(X.shape, y.shape)

# Split the data into training and testing (80% train, 20% test)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Build the model
model = Sequential([
    Flatten(input_shape=(100, 100, 3)),  # Flatten the 100x100x3 images
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(3, activation='softmax')  # 3 output classes: 10, 20, and 100
])

# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=16)

# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc}")

# Save the trained model
model.save('currency_note_model.h5')

# ---- Making Predictions ----
# For testing a new image
def predict_currency(img_path):
    img = cv2.imread(img_path)
    img = cv2.resize(img, (100, 100))  # Resize to 100x100
    img = img / 255.0  # Normalize
    img = np.expand_dims(img, axis=0)  # Add batch dimension
```

```
    # Predict the label
    prediction = model.predict(img)
    predicted_label = np.argmax(prediction)

    # Mapping the output back to denomination labels
    labels_map = {0: "10", 1: "20", 2: "100"}
    return labels_map[predicted_label]

# Example: Predict the currency note of a new image
img_path = "/content/20_new.jpg"  # Update with the path to your new image
predicted_denomination = predict_currency(img_path)

print(f"The predicted currency denomination is: {predicted_denomination}")
```

```
from keras.utils import plot_model
plot_model(model, to_file='model_architecture.png', show_shapes=True,
show_layer_names=True)
```

```
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()

model.add(Dense(3, activation='softmax'))  # For 3 classes
```

```
Accuracy plot
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
# Loss plot
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
=
```