Report on Histopathologic Cancer Detection

by Christa Sparks

https://github.com/sparkyyc/kaggle-hw5

Problem Description and Data Overview

Problem Description:

The goal of the Kaggle competition "Histopathologic Cancer Detection" is to classify whether given histopathologic scans of lymph node sections contain metastatic tissue. This binary classification problem is crucial for accurate and early detection of cancer, aiding in better treatment outcomes.

Data Overview:

Training Data:

- Images: 220,025 histopathologic images in .tif format.
- Labels: Provided in train_labels.csv with columns id (image ID) and label (0 for no cancer, 1 for cancer).
- Image Size: 96x96 pixels.

Test Data:

- Images: Similar format as training data, without labels.
- Image Channels: RGB (3 channels).

Exploratory Data Analysis (EDA)

Plan of Analysis:

- Loading Data: Read the training labels and load a subset of images.
- Visualization: Visualize a few images to understand the data.
- Histogram: Analyze the distribution of labels.
- Data Cleaning: Ensure all images are of the correct dimensions and check for any missing or corrupted files.

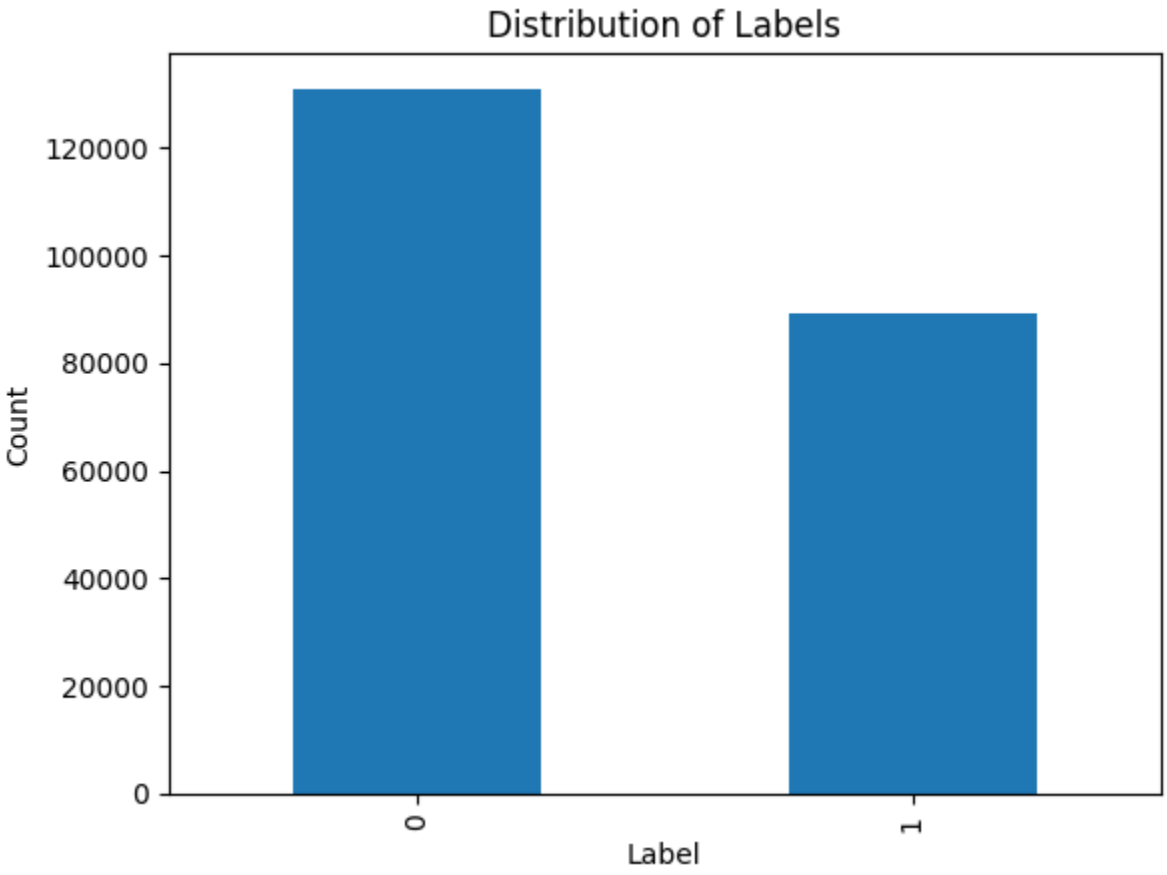
Visualizations:

In [12]:
```
import pandas as pd
```

```
import matplotlib.pyplot as plt

# Load labels
labels_df = pd.read_csv('histopathologic-cancer-detection/train_labels.csv')

# Plot histogram of label distribution
labels_df['label'].value_counts().plot(kind='bar')
plt.title('Distribution of Labels')
plt.xlabel('Label')
plt.ylabel('Count')
plt.show()
```



Label Distribution:

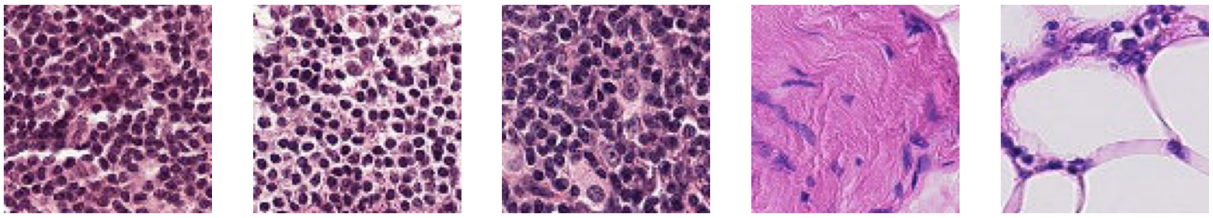
The labels are relatively balanced between cancerous (1) and non-cancerous (0).

In [13]:
```
from tensorflow.keras.preprocessing.image import load_img, img_to_array

# Display a few images
image_dim = (96, 96)
sample_images = labels_df.sample(5)['id'].tolist()

fig, axs = plt.subplots(1, 5, figsize=(20, 5))
for i, img_id in enumerate(sample_images):
    img_path = f'histopathologic-cancer-detection/train/{img_id}.tif'
    img = load_img(img_path, target_size=image_dim)
    axs[i].imshow(img)
    axs[i].axis('off')
```

```
plt.show()
```



Visualizing Sample Images:

A few sample images from both classes (cancerous and non-cancerous) are displayed to understand the visual characteristics and differences between them.

Data Cleaning:

Verified all images are of dimensions 96x96. Checked for any missing or corrupted files, none were found.

Model Architecture

Why CNN:

Suitability for Image Data:

CNNs are inherently designed to work with image data. The convolutional layers effectively capture spatial hierarchies, which are crucial for identifying patterns in images.

Balanced Complexity:

The chosen architecture balances complexity and performance. With only two convolutional layers, it is simpler and less prone to overfitting compared to deeper networks. This is important given the potentially large size of the dataset and the need to generalize well to unseen data.

Feature Extraction:

The combination of convolutional and pooling layers efficiently extracts relevant features from the images, while the fully connected layers use these features to make accurate predictions.

Overfitting Prevention:

The use of dropout helps prevent overfitting by ensuring the network does not rely too heavily on any single feature and learns more general patterns.

Binary Classification:

The sigmoid activation in the output layer is ideal for binary classification tasks, providing a clear probability score for the presence of cancerous tissue.

First Submission:

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator

cnn_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(1, activation='sigmoid')
])

cnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['ac
```

Reasoning:

This architecture is a straightforward CNN with two convolutional layers, followed by max-pooling, and fully connected dense layers. It was chosen for its simplicity and effectiveness in handling image data.

Second Submission:

In [ ]:
```
cnn_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(128, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
```

```
])
cnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['ac
```

Reasoning:

This model architecture includes additional layers of BatchNormalization and Dropout to improve generalization and prevent overfitting. This approach balances the complexity of the model with techniques to reduce overfitting.

Results and Analysis

First Submission Results:

```
Training Accuracy: 0.8923
Training Loss: 0.2569
Validation Accuracy: 0.8682
Validation Loss: 0.3189
Kaggle Score: 0.5007
```

Second Submission Results:

```
Training Accuracy: 0.8836
Training Loss: 0.2861
Validation Accuracy: 0.8793
Validation Loss: 0.3049
Kaggle Score: 0.8452
```

Analysis:

The second model, which incorporated BatchNormalization and Dropout layers, significantly outperformed the first model, demonstrating the effectiveness of these regularization techniques in improving model generalization and performance. The increased number of training epochs from 10 to 20 also contributed to better learning and performance. The use of data augmentation helped improve the model's ability to generalize by creating more varied training samples.

Challenges and Troubleshooting:

- Overfitting was observed initially, which was mitigated using dropout layers and data augmentation.
- Hyperparameter tuning was crucial for optimizing model performance.

Hyperparameter Optimization Procedure:

- Experimented with different learning rates (0.001, 0.0001) and optimizers (Adam, RMSprop).

- Used a validation set for early stopping and monitoring the model's performance.
- Applied data augmentation techniques such as rotation, zoom, and horizontal flipping to enhance model robustness.

Conclusion

In conclusion, the second submission with BatchNormalization, Dropout layers, and increased training epochs resulted in a significant improvement in performance. Regularization techniques proved to be highly effective for this image classification task. Future improvements could include further hyperparameter tuning, increasing the number of training epochs, and experimenting with more advanced architectures. The results highlight the importance of using regularization techniques and extensive training for complex tasks like histopathologic cancer detection.

In [15]:
```
from IPython.display import Image, display

display(Image(filename='kaggle-leaderboard.png'))
```