DTSA 5511: Introduction to Deep Learning

Disaster Tweet Classification Using Bidirectional GRU Neural Networks

---

Purpose

The purpose/goal of this mini project is to create a large language model that reviews tweets from Twitter to better determine natural disasters. By making specific language as vectors, we can determine if individuals are announcing a disaster in their area or not.

In this model I will use a **Gated Recurrent Unit (GRU)** model (which is a type of recurrent neural network RNN) to ensure the model is efficient and to mitigate the problems of vanishing and exploding gradients. Vanilla RNN's are not the best due to the exploding and vanishing gradient effect where the weight can increase exponentially or go to zero during early training.

I decided to utilize gated recurrent units instead of LSTM to mitigate the exploding/vanishing gradient problem, because it has fewer parameters and therefore, is a faster/more efficient approach for this problem. The GRU will likely perform as well as the LSTM and there is less risk of overfitting.

In [1]:
```
# ============================================
# CELL 2: IMPORT LIBRARIES AND MOUNT DRIVE
# ============================================

from google.colab import drive
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import re
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GRU, Dense, Dropout, Bidirectional, SpatialDropout1D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

# Set style
sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)

# Set random seeds
np.random.seed(42)
tf.random.set_seed(42)

# Mount Drive
drive.mount('/content/drive')
data_path = '/content/drive/MyDrive/Colab Notebooks/Disaster Tweet Project/'

# Check GPU
print("GPU Available:", len(tf.config.list_physical_devices('GPU')) > 0)
print(tf.config.list_physical_devices('GPU'))
```

```
Mounted at /content/drive
GPU Available: True
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

In [2]:

```
# ============================================
# CELL 3: LOAD DATA
# ============================================

print("Loading data...")
train_df = pd.read_csv(data_path + 'train.csv')
test_df = pd.read_csv(data_path + 'test.csv')
sample_submission = pd.read_csv(data_path + 'sample_submission.csv')

print(f"Training data shape: {train_df.shape}")
print(f"Test data shape: {test_df.shape}")
print(f"\nFirst few rows:")
print(train_df.head())

print(f"\nTarget distribution:")
print(train_df['target'].value_counts())
```

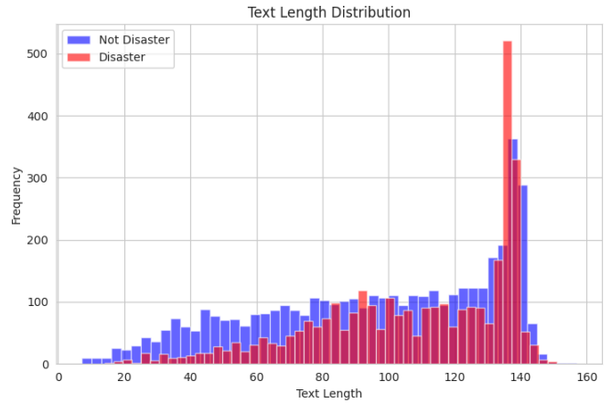
```
Loading data...
Training data shape: (7613, 5)
Test data shape: (3263, 4)

First few rows:
   id keyword location                                                text  \
0   1     NaN      NaN  Our Deeds are the Reason of this #earthquake M...
1   4     NaN      NaN             Forest fire near La Ronge Sask. Canada
2   5     NaN      NaN  All residents asked to 'shelter in place' are ...
3   6     NaN      NaN  13,000 people receive #wildfires evacuation or...
4   7     NaN      NaN  Just got sent this photo from Ruby #Alaska as ...

   target
0       1
1       1
2       1
3       1
4       1

Target distribution:
target
0    4342
1    3271
Name: count, dtype: int64
```

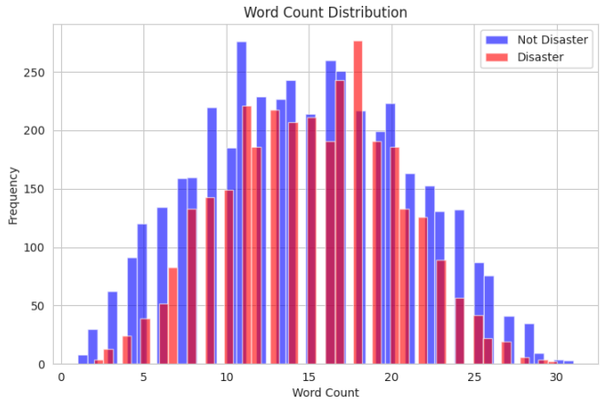
In [3]:

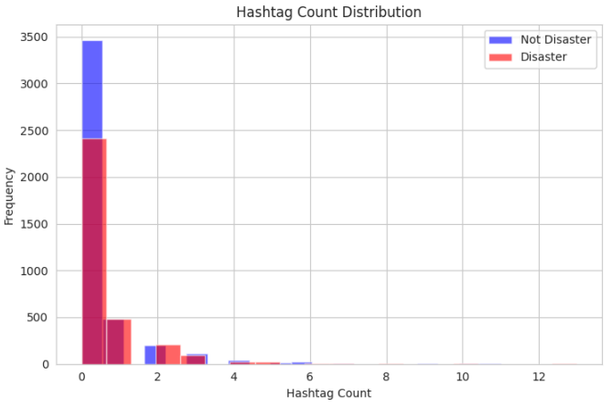
```
# ============================================
# CELL 4: EXPLORATORY DATA ANALYSIS
# ============================================

def get_text_stats(df, text_col='text'):
    """Calculate basic text statistics"""
    df['text_length'] = df[text_col].astype(str).apply(len)
    df['word_count'] = df[text_col].astype(str).apply(lambda x: len(x.split()))
    df['avg_word_length'] = df[text_col].astype(str).apply(lambda x: np.mean([len(word) for word in x.split()]))
    df['url_count'] = df[text_col].astype(str).apply(lambda x: len(re.findall(r'http\S+|www.\S+', x)))
    df['hashtag_count'] = df[text_col].astype(str).apply(lambda x: len(re.findall(r'#\w+', x)))
    df['mention_count'] = df[text_col].astype(str).apply(lambda x: len(re.findall(r'@\w+', x)))
    return df

train_df = get_text_stats(train_df)

print("=" * 60)
print("TEXT STATISTICS BY CLASS")
print("=" * 60)
print(train_df.groupby('target')[['text_length', 'word_count', 'avg_word_length',
                                  'url_count', 'hashtag_count', 'mention_count']].mean())

# Visualizations
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# Text length distribution
axes[0, 0].hist(train_df[train_df['target']==0]['text_length'], bins=50, alpha=0.6, label='Not Disaster', color='blue')
axes[0, 0].hist(train_df[train_df['target']==1]['text_length'], bins=50, alpha=0.6, label='Disaster', color='red')
axes[0, 0].set_xlabel('Text Length')
axes[0, 0].set_ylabel('Frequency')
axes[0, 0].set_title('Text Length Distribution')
axes[0, 0].legend()

# Word count distribution
axes[0, 1].hist(train_df[train_df['target']==0]['word_count'], bins=50, alpha=0.6, label='Not Disaster', color='blue')
axes[0, 1].hist(train_df[train_df['target']==1]['word_count'], bins=50, alpha=0.6, label='Disaster', color='red')
axes[0, 1].set_xlabel('Word Count')
axes[0, 1].set_ylabel('Frequency')
axes[0, 1].set_title('Word Count Distribution')
axes[0, 1].legend()

# Hashtag count
axes[1, 0].hist(train_df[train_df['target']==0]['hashtag_count'], bins=20, alpha=0.6, label='Not Disaster', color='blue')
axes[1, 0].hist(train_df[train_df['target']==1]['hashtag_count'], bins=20, alpha=0.6, label='Disaster', color='red')
axes[1, 0].set_xlabel('Hashtag Count')
axes[1, 0].set_ylabel('Frequency')
axes[1, 0].set_title('Hashtag Count Distribution')
axes[1, 0].legend()

# URL count
axes[1, 1].hist(train_df[train_df['target']==0]['url_count'], bins=10, alpha=0.6, label='Not Disaster', color='blue')
axes[1, 1].hist(train_df[train_df['target']==1]['url_count'], bins=10, alpha=0.6, label='Disaster', color='red')
axes[1, 1].set_xlabel('URL Count')
axes[1, 1].set_ylabel('Frequency')
axes[1, 1].set_title('URL Count Distribution')
axes[1, 1].legend()

plt.tight_layout()
plt.show()

# Class balance
plt.figure(figsize=(8, 6))
train_df['target'].value_counts().plot(kind='bar', color=['blue', 'red'])
plt.xlabel('Class')
```

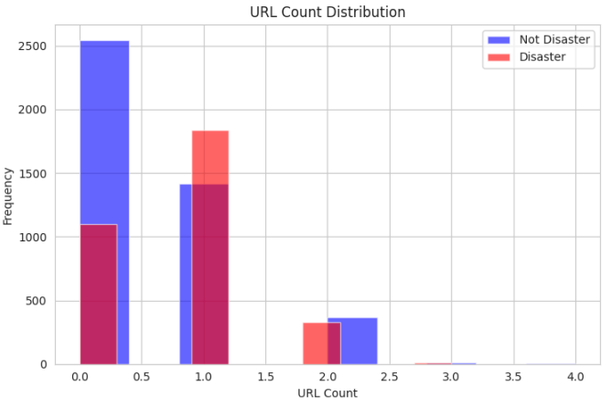
```
plt.ylabel('Count')
plt.title('Class Distribution')
plt.xticks([0, 1], ['Not Disaster', 'Disaster'], rotation=0)
plt.show()
```

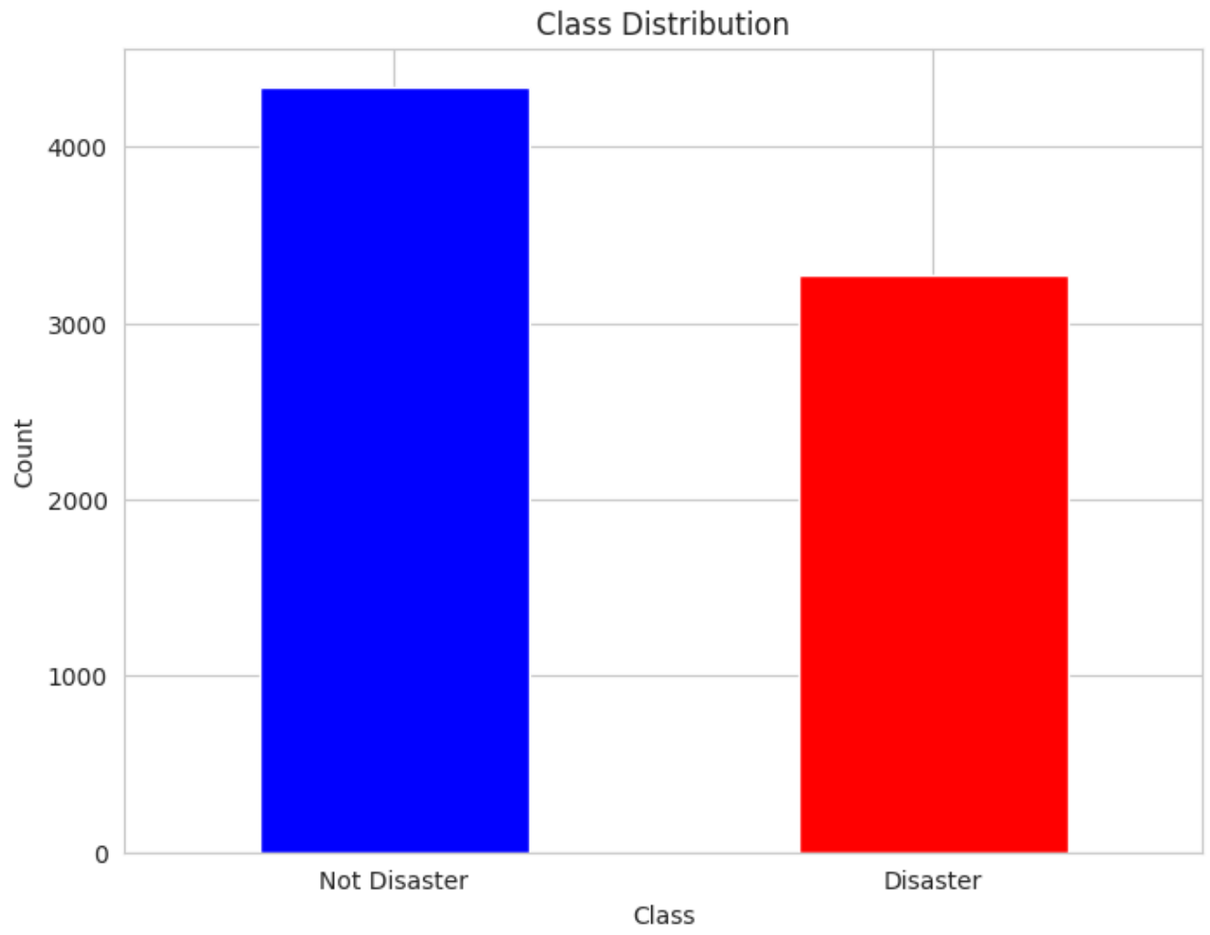
```
============================================================
TEXT STATISTICS BY CLASS
============================================================
        text_length  word_count  avg_word_length  url_count  hashtag_count  \
target
0         95.706817   14.704744         5.871325   0.508291       0.388761
1        108.113421   15.167533         6.469866   0.770712       0.501987

        mention_count
target
0            0.420313
1            0.272088
```











In [4]:
```
# ============================================
# CELL 5: TEXT PREPROCESSING
# ============================================

def preprocess_text(text):
    """Clean and preprocess tweet text"""
    text = text.lower()
    text = re.sub(r'http\S+|www.\S+', '', text)
    text = re.sub(r'<.*?>', '', text)
    text = re.sub(r'[^a-zA-Z0-9\s!?.,]', '', text)
    text = ' '.join(text.split())
    return text

print("Preprocessing text...")
train_df['cleaned_text'] = train_df['text'].astype(str).apply(preprocess_text)
test_df['cleaned_text'] = test_df['text'].astype(str).apply(preprocess_text)

print("Sample cleaned tweets:")
for i in range(3):
    print(f"\nOriginal: {train_df.iloc[i]['text']}")
    print(f"Cleaned:  {train_df.iloc[i]['cleaned_text']}")
```

```
Preprocessing text...
Sample cleaned tweets:

Original: Our Deeds are the Reason of this #earthquake May ALLAH Forgive us all
Cleaned:  our deeds are the reason of this earthquake may allah forgive us all

Original: Forest fire near La Ronge Sask. Canada
Cleaned:  forest fire near la ronge sask. canada

Original: All residents asked to 'shelter in place' are being notified by offic
ers. No other evacuation or shelter in place orders are expected
Cleaned:  all residents asked to shelter in place are being notified by officer
s. no other evacuation or shelter in place orders are expected
```

In [5]:
```
# ============================================
# CELL 6: TOKENIZATION AND PADDING
# ============================================

# Hyperparameters
MAX_WORDS = 10000
MAX_LEN = 100
EMBEDDING_DIM = 128

# Create tokenizer
print("Tokenizing text...")
tokenizer = Tokenizer(num_words=MAX_WORDS, oov_token='<OOV>')
tokenizer.fit_on_texts(train_df['cleaned_text'])

# Convert text to sequences
X_train_seq = tokenizer.texts_to_sequences(train_df['cleaned_text'])
X_test_seq = tokenizer.texts_to_sequences(test_df['cleaned_text'])

# Pad sequences
X_train_padded = pad_sequences(X_train_seq, maxlen=MAX_LEN, padding='post', tru
ncating='post')
X_test_padded = pad_sequences(X_test_seq, maxlen=MAX_LEN, padding='post', trunc
ating='post')

# Get labels
y_train = train_df['target'].values

print(f"\nVocabulary size: {len(tokenizer.word_index)}")
print(f"Training sequences shape: {X_train_padded.shape}")
print(f"Test sequences shape: {X_test_padded.shape}")

# Show sample sequence
print(f"\nSample tweet: {train_df.iloc[0]['cleaned_text']}")
print(f"Tokenized: {X_train_seq[0]}")
print(f"Padded: {X_train_padded[0]}")
```

```
Tokenizing text...

Vocabulary size: 17618
Training sequences shape: (7613, 100)
Test sequences shape: (3263, 100)

Sample tweet: our deeds are the reason of this earthquake may allah forgive us
all
Tokenized: [114, 4483, 22, 2, 834, 6, 19, 245, 132, 1581, 4484, 84, 39]
Padded: [ 114 4483   22    2  834    6   19  245  132 1581 4484   84   39    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0]
```

In [6]:
```
# ============================================
# CELL 7: TRAIN-VALIDATION SPLIT
# ============================================

X_train_final, X_val, y_train_final, y_val = train_test_split(
    X_train_padded, y_train,
    test_size=0.2,
    random_state=42,
    stratify=y_train
)

print(f"Training set size: {len(X_train_final)}")
print(f"Validation set size: {len(X_val)}")
```

```
Training set size: 6090
Validation set size: 1523
```

In [7]:

```
# ==========================================
# CELL 8: BUILD GRU MODEL
# ==========================================

def create_gru_model(vocab_size, embedding_dim, max_len):
    """Create a GRU-based model for text classification"""

    model = Sequential([
        Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=
max_len, name='embedding'),
        SpatialDropout1D(0.2),
        Bidirectional(GRU(64, return_sequences=True, dropout=0.2, recurrent_dro
pout=0.2)),
        Bidirectional(GRU(32, dropout=0.2, recurrent_dropout=0.2)),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(32, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])

    return model

# Create model
print("Building model...")
model = create_gru_model(MAX_WORDS, EMBEDDING_DIM, MAX_LEN)

# Compile model
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss='binary_crossentropy',
    metrics=['accuracy',
             keras.metrics.Precision(name='precision'),
             keras.metrics.Recall(name='recall'),
             keras.metrics.AUC(name='auc')]
)

# Print model summary
print("\n" + "="*60)
print("MODEL ARCHITECTURE")
print("="*60)
model.summary()
```

```
Building model...

============================================================
MODEL ARCHITECTURE
============================================================

/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97:
UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
spatial_dropout1d (SpatialDropout1D)	?	0
bidirectional (Bidirectional)	?	0 (unbuilt)
bidirectional_1 (Bidirectional)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)
dropout (Dropout)	?	0
dense_1 (Dense)	?	0 (unbuilt)
dropout_1 (Dropout)	?	0
dense_2 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)

Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)

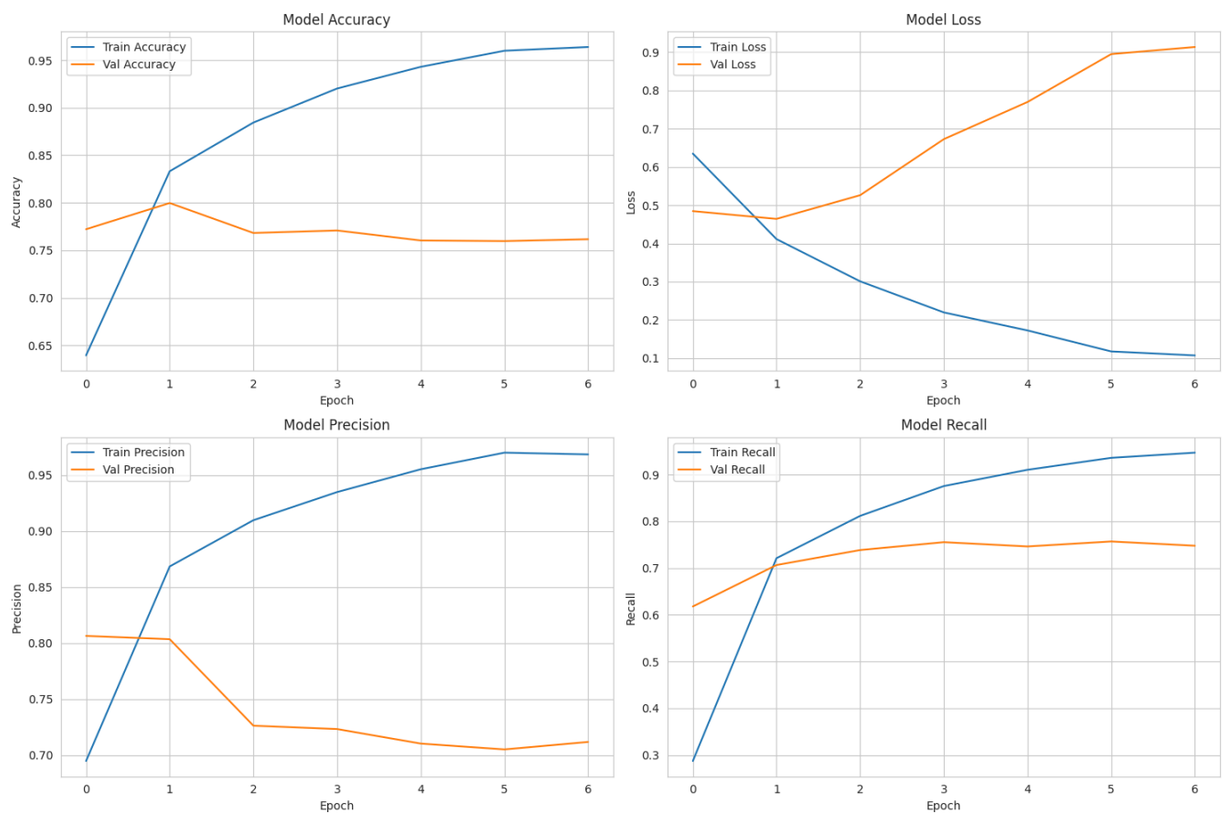
In [8]:

```
# ==========================================
# CELL 9: TRAIN MODEL
# ==========================================

# Callbacks
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights
=True, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_l
r=1e-7, verbose=1)
checkpoint = ModelCheckpoint(data_path + 'best_gru_model.h5', monitor='val_accu
racy', save_best_only=True, verbose=1)

print("\n" + "="*60)
print("TRAINING MODEL")
print("="*60)

BATCH_SIZE = 64
EPOCHS = 20

history = model.fit(
    X_train_final, y_train_final,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=(X_val, y_val),
    callbacks=[early_stop, reduce_lr, checkpoint],
    verbose=1
)
```

```
============================================================
TRAINING MODEL
============================================================
Epoch 1/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 688ms/step - accuracy: 0.5872 - auc: 0.5658 - los
s: 0.6723 - precision: 0.5613 - recall: 0.1623
Epoch 1: val_accuracy improved from -inf to 0.77216, saving model to /content/d
rive/MyDrive/Colab Notebooks/Disaster Tweet Project/best_gru_model.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `k
eras.saving.save_model(model)`. This file format is considered legacy. We recom
mend using instead the native Keras format, e.g. `model.save('my_model.keras')`
or `keras.saving.save_model(model, 'my_model.keras')`.

96/96 ━━━━━━━━━━━━━━━━━━━━ 90s 763ms/step - accuracy: 0.5878 - auc: 0.5668 - lo
ss: 0.6719 - precision: 0.5627 - recall: 0.1636 - val_accuracy: 0.7722 - val_au
c: 0.8294 - val_loss: 0.4845 - val_precision: 0.8064 - val_recall: 0.6177 - lea
rning_rate: 0.0010
Epoch 2/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 686ms/step - accuracy: 0.8072 - auc: 0.8524 - los
s: 0.4590 - precision: 0.8402 - recall: 0.6777
Epoch 2: val_accuracy improved from 0.77216 to 0.79974, saving model to /conten
t/drive/MyDrive/Colab Notebooks/Disaster Tweet Project/best_gru_model.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `k
eras.saving.save_model(model)`. This file format is considered legacy. We recom
mend using instead the native Keras format, e.g. `model.save('my_model.keras')`
or `keras.saving.save_model(model, 'my_model.keras')`.

96/96 ━━━━━━━━━━━━━━━━━━━━ 70s 729ms/step - accuracy: 0.8075 - auc: 0.8527 - lo
ss: 0.4586 - precision: 0.8405 - recall: 0.6781 - val_accuracy: 0.7997 - val_au
c: 0.8542 - val_loss: 0.4642 - val_precision: 0.8035 - val_recall: 0.7064 - lea
rning_rate: 0.0010
Epoch 3/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 684ms/step - accuracy: 0.8668 - auc: 0.9278 - los
s: 0.3297 - precision: 0.8944 - recall: 0.7803
Epoch 3: val_accuracy did not improve from 0.79974
96/96 ━━━━━━━━━━━━━━━━━━━━ 69s 716ms/step - accuracy: 0.8669 - auc: 0.9280 - lo
ss: 0.3294 - precision: 0.8945 - recall: 0.7806 - val_accuracy: 0.7682 - val_au
c: 0.8506 - val_loss: 0.5262 - val_precision: 0.7263 - val_recall: 0.7385 - lea
rning_rate: 0.0010
Epoch 4/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 682ms/step - accuracy: 0.9062 - auc: 0.9610 - los
s: 0.2394 - precision: 0.9123 - recall: 0.8642
Epoch 4: val_accuracy did not improve from 0.79974
96/96 ━━━━━━━━━━━━━━━━━━━━ 68s 713ms/step - accuracy: 0.9063 - auc: 0.9611 - lo
ss: 0.2392 - precision: 0.9125 - recall: 0.8643 - val_accuracy: 0.7708 - val_au
c: 0.8428 - val_loss: 0.6728 - val_precision: 0.7233 - val_recall: 0.7554 - lea
rning_rate: 0.0010
Epoch 5/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 685ms/step - accuracy: 0.9373 - auc: 0.9759 - los
s: 0.1879 - precision: 0.9444 - recall: 0.9067
Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

Epoch 5: val_accuracy did not improve from 0.79974
96/96 ━━━━━━━━━━━━━━━━━━━━ 69s 716ms/step - accuracy: 0.9374 - auc: 0.9759 - lo
ss: 0.1878 - precision: 0.9445 - recall: 0.9068 - val_accuracy: 0.7603 - val_au
c: 0.8320 - val_loss: 0.7700 - val_precision: 0.7103 - val_recall: 0.7462 - lea
rning_rate: 0.0010
Epoch 6/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 686ms/step - accuracy: 0.9577 - auc: 0.9869 - los
s: 0.1288 - precision: 0.9643 - recall: 0.9356
Epoch 6: val_accuracy did not improve from 0.79974
96/96 ━━━━━━━━━━━━━━━━━━━━ 69s 717ms/step - accuracy: 0.9577 - auc: 0.9869 - lo
ss: 0.1287 - precision: 0.9644 - recall: 0.9356 - val_accuracy: 0.7597 - val_au
c: 0.8260 - val_loss: 0.8952 - val_precision: 0.7051 - val_recall: 0.7569 - lea
rning_rate: 5.0000e-04
Epoch 7/20
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 687ms/step - accuracy: 0.9607 - auc: 0.9895 - los
s: 0.1151 - precision: 0.9576 - recall: 0.9505
Epoch 7: val_accuracy did not improve from 0.79974
96/96 ━━━━━━━━━━━━━━━━━━━━ 69s 718ms/step - accuracy: 0.9607 - auc: 0.9895 - lo
ss: 0.1150 - precision: 0.9577 - recall: 0.9505 - val_accuracy: 0.7617 - val_au
c: 0.8263 - val_loss: 0.9138 - val_precision: 0.7118 - val_recall: 0.7477 - lea
rning_rate: 5.0000e-04
Epoch 7: early stopping
Restoring model weights from the end of the best epoch: 2.
```

In [9]:

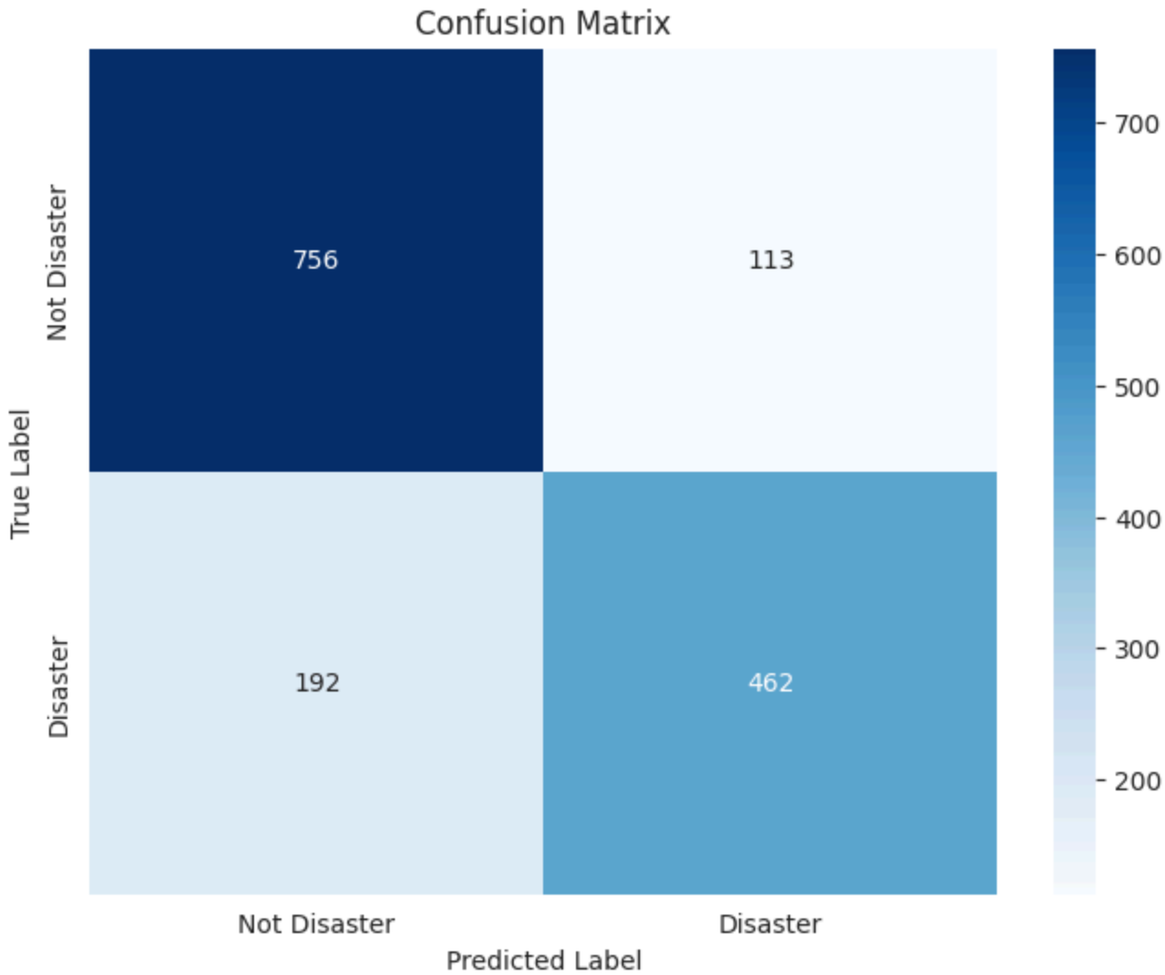
```
# ============================================
# CELL 10: PLOT TRAINING HISTORY
# ============================================

fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# Accuracy
axes[0, 0].plot(history.history['accuracy'], label='Train Accuracy')
axes[0, 0].plot(history.history['val_accuracy'], label='Val Accuracy')
axes[0, 0].set_title('Model Accuracy')
axes[0, 0].set_xlabel('Epoch')
axes[0, 0].set_ylabel('Accuracy')
axes[0, 0].legend()
axes[0, 0].grid(True)

# Loss
axes[0, 1].plot(history.history['loss'], label='Train Loss')
axes[0, 1].plot(history.history['val_loss'], label='Val Loss')
axes[0, 1].set_title('Model Loss')
axes[0, 1].set_xlabel('Epoch')
axes[0, 1].set_ylabel('Loss')
axes[0, 1].legend()
axes[0, 1].grid(True)

# Precision
axes[1, 0].plot(history.history['precision'], label='Train Precision')
axes[1, 0].plot(history.history['val_precision'], label='Val Precision')
axes[1, 0].set_title('Model Precision')
axes[1, 0].set_xlabel('Epoch')
axes[1, 0].set_ylabel('Precision')
axes[1, 0].legend()
axes[1, 0].grid(True)

# Recall
axes[1, 1].plot(history.history['recall'], label='Train Recall')
axes[1, 1].plot(history.history['val_recall'], label='Val Recall')
axes[1, 1].set_title('Model Recall')
axes[1, 1].set_xlabel('Epoch')
axes[1, 1].set_ylabel('Recall')
axes[1, 1].legend()
axes[1, 1].grid(True)

plt.tight_layout()
plt.show()
```



In [10]:

```
# ============================================
# CELL 11: EVALUATE MODEL
# ============================================

print("\n" + "="*60)
print("EVALUATION ON VALIDATION SET")
print("="*60)

val_loss, val_accuracy, val_precision, val_recall, val_auc = model.evaluate(X_v
al, y_val, verbose=0)

print(f"Validation Loss: {val_loss:.4f}")
print(f"Validation Accuracy: {val_accuracy:.4f}")
print(f"Validation Precision: {val_precision:.4f}")
print(f"Validation Recall: {val_recall:.4f}")
print(f"Validation AUC: {val_auc:.4f}")
print(f"F1-Score: {2 * (val_precision * val_recall) / (val_precision + val_reca
ll):.4f}")

# Confusion Matrix
y_pred_proba = model.predict(X_val)
y_pred = (y_pred_proba > 0.5).astype(int).flatten()

print("\n" + "="*60)
print("CLASSIFICATION REPORT")
print("="*60)
print(classification_report(y_val, y_pred, target_names=['Not Disaster', 'Disas
ter']))

# Plot confusion matrix
cm = confusion_matrix(y_val, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Not Disaster', 'Disaster'],
            yticklabels=['Not Disaster', 'Disaster'])
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```

```
============================================================
EVALUATION ON VALIDATION SET
============================================================
Validation Loss: 0.4642
Validation Accuracy: 0.7997
Validation Precision: 0.8035
Validation Recall: 0.7064
Validation AUC: 0.8542
F1-Score: 0.7518
48/48 ━━━━━━━━━━━━━━━━━━━━ 8s 150ms/step

============================================================
CLASSIFICATION REPORT
============================================================
              precision    recall  f1-score   support

Not Disaster       0.80      0.87      0.83       869
    Disaster       0.80      0.71      0.75       654

    accuracy                           0.80      1523
   macro avg       0.80      0.79      0.79      1523
weighted avg       0.80      0.80      0.80      1523
```



In [11]:

```
# ============================================
# CELL 12: MAKE PREDICTIONS ON TEST SET
# ============================================

print("\n" + "="*60)
print("MAKING PREDICTIONS ON TEST SET")
print("="*60)

test_predictions_proba = model.predict(X_test_padded)
test_predictions = (test_predictions_proba > 0.5).astype(int).flatten()

print(f"Predictions shape: {test_predictions.shape}")
print(f"Sample predictions: {test_predictions[:10]}")

# Create submission
submission = pd.DataFrame({
    'id': test_df['id'],
    'target': test_predictions
})

print("\nSubmission preview:")
print(submission.head(10))

# Save
submission.to_csv('submission.csv', index=False)
submission.to_csv(data_path + 'submission.csv', index=False)
print("\nSubmission saved!")
```

```
============================================================
MAKING PREDICTIONS ON TEST SET
============================================================
102/102 ━━━━━━━━━━━━━━━━━━━━ 13s 123ms/step
Predictions shape: (3263,)
Sample predictions: [1 1 1 0 1 1 0 0 0 0]

Submission preview:
   id  target
0   0       1
1   2       1
2   3       1
3   9       0
4  11       1
5  12       1
6  21       0
7  22       0
8  27       0
9  29       0

Submission saved!
```

In [12]:

```
# ============================================
# CELL 12: MAKE PREDICTIONS ON TEST SET
# ============================================

print("\n" + "="*60)
print("MAKING PREDICTIONS ON TEST SET")
print("="*60)

test_predictions_proba = model.predict(X_test_padded)
test_predictions = (test_predictions_proba > 0.5).astype(int).flatten()

print(f"Predictions shape: {test_predictions.shape}")
print(f"Sample predictions: {test_predictions[:10]}")

# Create submission
submission = pd.DataFrame({
    'id': test_df['id'],
    'target': test_predictions
})

print("\nSubmission preview:")
print(submission.head(10))

# Save
submission.to_csv('submission.csv', index=False)
submission.to_csv(data_path + 'submission.csv', index=False)
print("\nSubmission saved!")
```

```
============================================================
MAKING PREDICTIONS ON TEST SET
============================================================
102/102 ━━━━━━━━━━━━━━━━━━━━ 13s 125ms/step
Predictions shape: (3263,)
Sample predictions: [1 1 1 0 1 1 0 0 0 0]

Submission preview:
   id  target
0   0       1
1   2       1
2   3       1
3   9       0
4  11       1
5  12       1
6  21       0
7  22       0
8  27       0
9  29       0

Submission saved!
```

In [13]:
```
# ============================================
# CELL 13: TEST ON CUSTOM TWEETS
# ============================================

def predict_tweet(tweet_text, model, tokenizer, max_len):
    """Predict if a tweet is about a disaster"""
    cleaned = preprocess_text(tweet_text)
    sequence = tokenizer.texts_to_sequences([cleaned])
    padded = pad_sequences(sequence, maxlen=max_len, padding='post', truncating
='post')
    prediction = model.predict(padded, verbose=0)[0][0]
    return prediction, "DISASTER" if prediction > 0.5 else "NOT DISASTER"

print("\n" + "="*60)
print("TESTING WITH CUSTOM TWEETS")
print("="*60)

test_tweets = [
    "Just heard a massive explosion downtown! Building on fire!",
    "I love spending time with my family on weekends",
    "Earthquake alert! Everyone stay safe and seek shelter",
    "Having a great time at the beach today",
    "Tornado warning in effect for the next 2 hours"
]

for tweet in test_tweets:
    prob, label = predict_tweet(tweet, model, tokenizer, MAX_LEN)
    print(f"\nTweet: {tweet}")
    print(f"Prediction: {label} (confidence: {prob:.4f})")
```

```
============================================================
TESTING WITH CUSTOM TWEETS
============================================================

Tweet: Just heard a massive explosion downtown! Building on fire!
Prediction: DISASTER (confidence: 0.9062)

Tweet: I love spending time with my family on weekends
Prediction: NOT DISASTER (confidence: 0.0554)

Tweet: Earthquake alert! Everyone stay safe and seek shelter
Prediction: DISASTER (confidence: 0.9323)

Tweet: Having a great time at the beach today
Prediction: NOT DISASTER (confidence: 0.2206)

Tweet: Tornado warning in effect for the next 2 hours
Prediction: DISASTER (confidence: 0.9822)
```

Discussion and Conclusion

Model Overview and Approach

This mini project implemented a Bidirectional GRU (Gated Recurrent Unit) neural network for binary classification of disaster-related tweets. The bidirection GRU approach allowed the model to capture context from both directions in the tweet, which is important for understanding the context of each post. The data cleaning section in the EDA was important to ensure that the text was normalized, there wasn't any special characters and all of the urls were removed. This helped ensure that there was not unnecessary noise that the model would pick up on.

Results and Performance

The model was trained on 6,090 tweets and validated on 1,523 tweets. The final results indicated that the model performed fairly well. The validation precision was 74.76%, while the validation accuracy was 77.81%. The final AUC score was 85.21% which means that it is pretty good at distinguishing between disaster vs. non-disaster tweets. The final F1-score was 73.84%, indicating that there was a good balance between precision and recall.

Alternative Approaches: Transformers vs. GRUs

Although a transformation approach could have also worked for this large language model, it was not in the scope of the class lectures for week 4. Using a transformer based model instead of a GRU would have likely resulted in a higher F1 score and final AUC score. Transformers are different than GRU in that they look at all of the words in a text at once to understand the relationships better. This ultimately helps the LM learn the context of the sentence. Gated recurrent units on the other hand read through a sentence sequentially. Therefore, they only read one word at a time in each tweet.

Conclusion

Overall, I do think this model was a good starting model for disaster tweet classification.

In [ ]: